图灵程序设计丛书



[英] Dr. M. O. Faruque Sarker 著　安道 译

Python 网络编程攻略

Python Network Programming Cookbook

人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

M. O. Faruque Sarker博士

软件架构师，开发运维工程师，最近几年主持开发了很多Python软件项目。他是多种开源技术方面的专家，对于在线学习平台、Web应用平台、敏捷软件开发等技术有深入的研究。从1999年开始，他一直在部署Linux和开源软件。他应邀参与了2009年和2010年的谷歌编程之夏，为开源项目BlueZ和Tahoe-LAFS做出了贡献。目前他在英国伦敦大学学院工作，研究方向是自组织云架构。

安道

人子人夫人父，机械工程师，翻译爱好者，偶尔写代码。

图灵程序设计丛书



[英] Dr. M. O. Faruque Sarker 著　安道 译

Python 网络编程攻略

Python Network Programming Cookbook

人民邮电出版社
北　京

图书在版编目（CIP）数据

Python网络编程攻略 / （英）萨卡尔（Sarker,D.M.O.F.）著；安道译. -- 北京：人民邮电出版社，2014.12
（图灵程序设计丛书）
ISBN 978-7-115-37269-7

Ⅰ. ①P… Ⅱ. ①萨… ②安… Ⅲ. ①软件工具—程序设计 Ⅳ. ①TP311.56

中国版本图书馆CIP数据核字(2014)第232396号

内 容 提 要

本书介绍了现实世界中各种网络任务的真实示例，通过 70 多篇攻略讨论了 Python 网络编程的高阶话题，包括编写简单的网络客户端和服务器、HTTP 协议网络编程、跨设备编程、屏幕抓取以及网络安全监控，等等。本书可以作为任何一门网络编程课程中培养实践技能的补充材料。

本书适合网络程序员、系统 / 网络管理员及 Web 应用程序开发人员阅读。

◆ 著　　　[英] Dr. M. O. Faruque Sarker
译　　　安　道
责任编辑　岳新欣
责任印制　杨林杰

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　　　印刷

◆ 开本：800×1000　1/16
印张：11.75
字数：255千字　　　　2014年 12 月第 1 版
印数：1 – 3 500册　　　　2014年 12 月北京第 1 次印刷

著作权合同登记号　图字：01-2014-6530号

定价：45.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京崇工商广字第 0021 号

版权声明

致　　谢

我要感谢所有为本书出版做出贡献的人，包括出版社、技术审阅人员、编辑、好友和我的家人，尤其是我的妻子Shahinur Rijuani，她给了我关爱，也支持我的工作。我还要感谢耐心等待本书出版的读者，以及给予我很多重要反馈的读者。

前　　言

很高兴看到本书出版了，我要感谢所有为本书的出版做出贡献的人。本书是Python网络编程方面的探索性指南，涉及了很多网络协议，例如TCP/UDP、HTTP/HTTPS、FTP、SMTP、POP3、IMAP、CGI等。Python功能强大且具交互性，用它来开发解决实际问题的脚本是一种享受，比如处理网络和系统管理操作、开发Web应用、与本地和远程网络交互、捕获并分析低层网络数据包，等等。本书的主要目的是教你动手完成这些任务，因此不会涉及太多理论，而是注重实践。

写作本书的过程中我一直记着要遵守“开发运维”的理念，开发者或多或少都要负责一些运维，即部署应用程序以及管理它的方方面面，例如管理远程服务器、监控、扩放以及性能优化等。书中用到了很多第三方开源Python库，有效解决了多种不同的问题。其中很多库我每天都用，通过它们自动化运行开发和运维任务简直是一种享受。例如，我使用Fabric自动完成软件开发过程中的任务。其他库也各有各的用处，例如搜索互联网、屏幕抓取、在Python脚本中发送电子邮件。

希望你能从本书的攻略中受益，并根据需求扩展它们，让其功能更强大，用起来更得心应手。

本书内容

第1章“套接字、IPv4和简单的客户端/服务器编程”通过多个小型任务讲解Python的核心网络库，教你开发一个客户端/服务器程序。

第2章“使用多路复用套接字I/O提升性能”讨论很多使用内置库和第三方库扩放客户端/服务器程序的实用技术。

第3章“IPv6、Unix域套接字和网络接口”主要关注本地设备的管理和本地网络的维护。

第4章“HTTP协议网络编程”开发一个多功能迷你命令行浏览器，可以提交表单、处理cookie、管理分段下载、压缩数据，还能通过HTTPS交付安全内容。

第5章“电子邮件协议、FTP和CGI编程”带你一起体验自动处理FTP和电子邮件相关任务的乐趣，例如管理Gmail账户、使用脚本收发邮件，还要为Web应用开发一个留言板。

第6章“屏幕抓取和其他实用程序”介绍如何使用多个第三方Python库实现一些实际的任务，

例如在谷歌地图上找到公司的位置、从维基百科中抓取信息、在GitHub中搜索代码仓库，以及从BBC读取新闻。

第7章“跨设备编程”带你体验如何使用SSH自动执行系统管理和部署任务。使用SSH，在你的笔记本电脑上就可以远程执行命令、安装包，或者架设新网站。

第8章“使用Web服务：XML-RPC、SOAP和REST”介绍不同的API协议，例如XML-RPC、SOAP和REST。使用这些协议可以通过编程的方式从任何网站或Web服务中读取信息，或者与之交互。例如，可以在亚马逊或谷歌中搜索商品。

第9章“网络监控和安全性”介绍捕获、存储、分析和处理网络数据包的多种技术。了解这些技术之后，你就能使用简洁的Python脚本分析并解决网络安全问题。

阅读本书前的准备工作

你要有一个可以使用的个人电脑或者笔记本电脑，最好安装了某种现代Linux操作系统，例如 Ubuntu、Debian或CentOS等。书中大部分攻略也能在其他平台上运行，例如Windows和Mac OS。

你还需要连接互联网，以便安装攻略中提到的第三方软件库。如果不方便上网，可以下载所有第三方库，一次性安装好。

下面列出本书使用的第三方库及其下载地址。

- **ntplib**：https://pypi.python.org/pypi/ntplib/
- **diesel**：https://pypi.python.org/pypi/diesel/
- **nmap**：https://pypi.python.org/pypi/python-nmap
- **scapy**：https://pypi.python.org/pypi/scapy
- **netifaces**：https://pypi.python.org/pypi/netifaces/
- **netaddr**：https://pypi.python.org/pypi/netaddr
- **pyopenssl**：https://pypi.python.org/pypi/pyOpenSSL
- **pygeocoder**：https://pypi.python.org/pypi/pygocoder
- **pyyaml**：https://pypi.python.org/pypi/PyYAML
- **requests**：https://pypi.python.org/pypi/requests

- **feedparser**：https://pypi.python.org/pypi/feedparser
- **paramiko**：https://pypi.python.org/pypi/paramiko/
- **fabric**：https://pypi.python.org/pypi/Fabric
- **supervisor**：https://pypi.python.org/pypi/supervisor
- **xmlrpclib**：https://pypi.python.org/pypi/xmlrpclib
- **SOAPpy**：https://pypi.python.org/pypi/SOAPpy
- **bottlenose**：https://pypi.python.org/pypi/bottlenose
- **construct**：https://pypi.python.org/pypi/construct/

运行某些攻略还要用到一些非Python软件，如下所示。

- **postfix**：http://www.postfix.org/
- **OpenSSH服务器**：http://www.openssh.com/
- **MySQL服务器**：http://downloads.mysql.com/
- **Apache2**：http://httpd.apache.org/download.cgi

本书读者

如果你是网络程序员、系统/网络管理员或者Web程序开发者，本书是理想之选。你应该对Python编程语言和TCP/IP的概念有个基本的了解。不过，对初学者来说，在阅读本书的过程中也能加强对这些概念的理解。本书也可作为网络编程课程的参考材料，用来培养实践操作能力。

排版约定

阅读本书时你会发现不同类别的信息使用了不同的文本样式，下面举例说明其中一些样式，及其表示的含义。

文本中的代码、数据库表名、文件扩展名和用户输入使用下述方式表示：

如果想知道远程设备的IP地址，可以使用内置的库函数`gethostbyname()`。

代码块的表示方法如下：

```
def test_socket_timeout():
```

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print "Default socket timeout: %s" %s.gettimeout()
s.settimeout(100)
print "Current socket timeout: %s" %s.gettimeout()
```

命令行输入和输出的表示方法如下：

```
$ python 2_5_echo_server_with_diesel.py --port=8800
[2013/04/08 11:48:32] {diesel} WARNING:Starting diesel <hand-rolled select.epoll>
```

读者反馈

我们始终期待收到读者的反馈。请让我们知道你对这本书的看法，喜欢哪些内容，不喜欢哪些内容。读者的反馈对我们来说十分重要，这样我们才能出版读者最需要的图书。

常规反馈请通过电子邮件发到feedback@packtpub.com，在邮件主题中请注明书名。

如果你是某方面的专家，有兴趣写一本书，或者想为其他书做贡献，请阅读我们的作者指南，地址是www.packtpub.com/authors。

客户支持

现在你已经拥有了一本由Packt出版的书，为了让你的付出得到最大回报，我们还为你提供了其他方面的服务。

下载示例代码

如果你是通过http://www.packtpub.com的注册账户购买的图书，可以从该账户中下载相应Packt图书的示例代码。如果你是从其他地方购买的本书，可以访问http://www.packtpub.com/support，注册账户后，我们将会为你发送一封附有示例代码文件的电子邮件。

勘误

虽然我们会全力确保书中内容的准确性，但错误仍在所难免。如果你在某本书中发现了错误（文字错误或代码错误），而且愿意向我们提交这些错误，我们感激不尽。这样不仅可以消除其他读者的疑虑，也有助于改进后续版本。若想提交你发现的错误，请访问http://www.packtpub.com/submit-errata，在“Errata Submission Form”（提交勘误表单）中选择相应图书，输入勘误详情。勘误通过验证之后将上传到Packt网站，或添加到现有的勘误列表中。若想查看某本书的现有勘误信息，请访问http://www.packtpub.com/support，选择相应的书名。

举报盗版

对所有媒体来说，互联网盗版都是一个棘手的问题。Packt很重视版权保护。如果你在互联网上发现我们公司出版物的任何非法复制品，请及时告知我们网址或网站名称，以便我们采取补救措施。

如果发现可疑盗版材料，请通过copyright@packtpub.com联系我们。

你的举报可以帮助我们保护作者权益，也有利于我们不断出版高品质的图书。我们对你深表感激。

疑难解答

如果你对本书的任何内容存有疑问，请发送电子邮件到questions@packtpub.com，我们会尽力解决。

目　　录

第 1 章 套接字、IPv4和简单的客户端/服务器编程

本章攻略：

- ❑ 打印设备名和IPv4地址
- ❑ 获取远程设备的IP地址
- ❑ 将IPv4地址转换成不同的格式
- ❑ 通过指定的端口和协议找到服务名
- ❑ 主机字节序和网络字节序之间相互转换
- ❑ 设定并获取默认的套接字超时时间
- ❑ 优雅地处理套接字错误
- ❑ 修改套接字发送和接收的缓冲区大小
- ❑ 把套接字改成阻塞或非阻塞模式
- ❑ 重用套接字地址
- ❑ 从网络时间服务器上获取并打印当前时间
- ❑ 编写一个SNTP客户端
- ❑ 编写一个简单的回显客户端/服务器应用

1.1 简介

本章通过一些简单的攻略介绍Python的核心网络库。Python的`socket`模块提供了类方法和实例方法，二者的区别在于使用类方法时不需要创建套接字对象实例。这是一种很直观的方法。例如，打印设备的IP地址不需要创建套接字对象，而只需调用套接字的类方法。但是，如果要把数据发送给服务器程序，那么创建一个套接字对象来处理具体的操作则更加自然。本章介绍的攻略可以分成如下三类：

- ❑ 前几个攻略使用类方法获取关于主机、网络以及目标服务的有用信息；

- 随后的几个攻略使用实例方法，演示了常用的套接字操作，例如处理套接字超时、缓冲区大小和阻塞模式等；
- 最后，结合使用类方法和实例方法开发客户端，执行一些实际的任务，例如使设备时间与网络服务器同步，编写通用的客户端/服务器脚本。

你可以使用本章演示的方法编写自己的客户端/服务器应用。

1.2　打印设备名和 IPv4 地址

有时，你需要快速查看设备的某些信息，例如主机名、IP地址和网络接口的数量等。这些信息使用Python脚本很容易获取。

1.2.1　准备工作

编写代码之前先要在设备上安装Python。大多数Linux发行版都预装了Python。如果使用微软Windows操作系统，可以从Python的网站上下载二进制文件：http://www.python.org/download/。

要了解系统是否已经安装了Python，可以查阅操作系统的文档。在设备上安装好Python之后，可以在命令行中输入`python`，尝试打开Python解释器。输入`python`后应该显示解释器提示符`>>>`，具体的输出如下所示：

```
~$ python
Python 2.7.1+ (r271:86832, Apr 11 2011, 18:05:24)
[GCC 4.5.2] on linux2
Type "help", "copyright", "credits" or "license" for more information. >>>
```

1.2.2　实战演练

这个攻略很简短，可以直接写在Python解释器中。

首先，使用下面的命令导入Python中的`socket`库：

```
>>> import socket
```

然后，调用`socket`库提供的`gethostname()`方法，把结果保存在一个变量中，如下所示：

```
>>> host_name = socket.gethostname()
>>> print "Host name: %s" %host_name
Host name: debian6
>>> print "IP address: %s" %socket.gethostbyname(host_name)
IP address: 127.0.1.1
```

这些操作可以使用内置的类方法，定义成一个独立的函数`print_machine_info()`。

我们要在常用的`__main__`代码块中调用这个函数。运行时，Python会为某些内部变量赋值，例如`__name__`。在这里，`__name__`表示调用程序的进程名。如果在命令行中运行脚本（如后面的命令所示），`__name__`的值是`__main__`。但是，如果在其他脚本中导入，情况就不同了。也就是说，如果在命令行中调用这个模块，会自动运行`print_machine_info()`函数；如果在其他脚本中导入，用户就要手动调用这个函数。

代码清单1-1展示了如何获取设备的信息，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter -1
# This program is optimized for Python 2.7. It may run on any
# other Python version with/without modifications.

import socket

def print_machine_info():
    host_name = socket.gethostname()
    ip_address = socket.gethostbyname(host_name)
    print "Host name: %s" % host_name
    print "IP address: %s" % ip_address

if __name__ == '__main__':
    print_machine_info()
```

若想运行这个脚本，要在命令行中指定源码文件，如下所示：

```
$ python 1_1_local_machine_info.py
```

在我的设备上，显示了如下输出：

```
Host name: debian6
IP address: 127.0.0.1
```

在你的设备上，输出的内容根据系统的主机配置会有所不同。

1.2.3　原理分析

`import socket`语句导入Python提供的一个核心网络库。然后调用两个工具函数：`gethostname()`和`gethostbyname(host_name)`。在命令行中可以输入`help(socket.gethostname)`查看帮助信息，或者在浏览器中访问http://docs.python.org/3/library/socket.html。在命令行中查看这两个函数的帮助信息，得到的输出如下：

```
gethostname(...)
    gethostname() -> string
    Return the current host name.

gethostbyname(...)
```

```
gethostbyname(host) -> address
 Return the IP address (a string of the form '255.255.255.255') for a host.
```

第一个函数没有参数，返回所在主机或本地主机的名字。第二个函数接收一个参数hostname，返回对应的IP地址。

1.3　获取远程设备的 IP 地址

有时需要把设备的主机名转换成对应的IP地址，例如快速查询域名。本攻略介绍一个简单的函数来完成这一操作。

1.3.1　实战演练

如果想知道远程设备的IP地址，可以使用内置的库函数gethostbyname()，其参数是远程设备的主机名。

这里，我们要调用的是类函数gethostbyname()。让我们来看一下这个简短的代码片段。

代码清单1-2展示了如何获取远程设备的IP地址，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 1
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import socket

def get_remote_machine_info():
    remote_host = 'www.python.org'
    try:
        print "IP address: %s" %socket.gethostbyname(remote_host)
    except socket.error, err_msg:
        print "%s: %s" %(remote_host, err_msg)

if __name__ == '__main__':
    get_remote_machine_info()
```

运行上述代码会得到以下输出：

```
$ python 1_2_remote_machine_info.py
IP address of www.python.org: 82.94.164.162
```

1.3.2　原理分析

这个攻略把gethostbyname()方法包装在用户定义的get_remote_machine_info()函数中，还引入了异常处理的概念。如上述代码所示，我们把主要的函数调用放在try-except块

中，这就意味着，如果执行函数gethostbyname()的过程中发生了错误，这个错误将由try-except块处理。

假如我们修改remote_host参数的值，把www.python.org改成一个不存在的域名，例如www.pytgo.org，然后执行下述命令：

```
$ python 1_2_remote_machine_info.py
www.pytgo.org: [Errno -5] No address associated with hostname
```

try-except块捕获了错误，并向用户显示了一个错误消息，说明域名www.pytgo.org没有对应的IP地址。

1.4 将 IPv4 地址转换成不同的格式

如果要使用低层网络函数，有时普通的字符串形式的IP地址并不是很有用，需要把它们转换成打包后的32位二进制格式。

1.4.1 实战演练

Python的socket库提供了很多用来处理不同IP地址格式的函数，这里我们使用其中的两个：inet_aton()和inet_ntoa()。

我们来定义convert_ip4_address()函数，调用inet_aton()和inet_ntoa()转换IP地址。我们要使用两个示例IP地址：127.0.0.1和192.168.0.1。

代码清单1-3展示了如何定义convert_ip4_address()函数，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 1
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import socket
from binascii import hexlify

def convert_ip4_address():
    for ip_addr in ['127.0.0.1', '192.168.0.1']:
        packed_ip_addr = socket.inet_aton(ip_addr)
        unpacked_ip_addr = socket.inet_ntoa(packed_ip_addr)
        print "IP Address: %s => Packed: %s, Unpacked: %s"\
            %(ip_addr, hexlify(packed_ip_addr), unpacked_ip_addr)

if __name__ == '__main__':
    convert_ip4_address()
```

现在，运行这个攻略，会看到以下输出：

```
$ python 1_3_ip4_address_conversion.py

IP Address: 127.0.0.1 => Packed: 7f000001, Unpacked: 127.0.0.1
IP Address: 192.168.0.1 => Packed: c0a80001, Unpacked: 192.168.0.1
```

1.4.2　原理分析

在这个攻略中，使用for-in语句把两个字符串形式的IP地址转换成打包后的32位二进制格式，而且还调用了binascii模块中的hexlify函数，以十六进制形式表示二进制数据。

1.5　通过指定的端口和协议找到服务名

如果想找到网络服务，最好知道该服务运行在TCP或UDP协议的哪个端口上。

1.5.1　准备工作

如果知道网络服务使用的端口，可以调用socket库中的getservbyport()函数来获取服务的名字。调用这个函数时可以根据情况决定是否提供协议名。

1.5.2　实战演练

我们来定义find_service_name()函数，在Python的for-in循环中调用函数getservbyport()，解析几个端口，例如80和25。

代码清单1-4展示了如何定义find_service_name()函数，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter -  1
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import socket

def find_service_name():
    protocolname = 'tcp'
    for port in [80, 25]:
        print "Port: %s => service name: %s" %(port, socket.getservbyport(port,
protocolname))
    print "Port: %s => service name: %s" %(53, socket.getservbyport(53, 'udp'))

if __name__ == '__main__':
    find_service_name()
```

运行这个脚本，会看到如下输出：

```
$ python 1_4_finding_service_name.py

Port: 80 => service name: http
Port: 25 => service name: smtp
Port: 53 => service name: domain
```

1.5.3　原理分析

在这个攻略中，使用for-in语句遍历一组变量。在每次遍历中，获取端口对应的服务名。

1.6　主机字节序和网络字节序之间相互转换

编写低层网络应用时，或许需要处理通过电缆在两台设备之间传送的低层数据。在这种操作中，需要把主机操作系统发出的数据转换成网络格式，或者做逆向转换，因为这两种数据的表示方式不一样。

1.6.1　实战演练

Python的socket库提供了将数据在网络字节序和主机字节序之间相互转换的函数。你可能想了解这些函数，例如ntohl()和htonl()。

我们来定义convert_integer()函数，调用ntohl()和htonl()类函数来转换不同格式的数据。

代码清单1-5展示了如何定义convert_integer()函数，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter -1
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import socket

def convert_integer():
    data = 1234
    # 32-bit
    print "Original: %s => Long host byte order: %s, Network byte order: %s"\
        %(data, socket.ntohl(data), socket.htonl(data))
    # 16-bit
    print "Original: %s => Short host byte order: %s, Network byte order: %s"\
        %(data, socket.ntohs(data), socket.htons(data))

if __name__ == '__main__':
    convert_integer()
```

运行这个攻略，会看到以下输出：

```
$ python 1_5_integer_conversion.py
Original: 1234 => Long host byte order: 3523477504, Network byte order: 3523477504
Original: 1234 => Short host byte order: 53764, Network byte order: 53764
```

1.6.2　原理分析

在这个攻略中，我们以整数为例，演示了如何把它转换成网络字节序和主机字节序。socket库中的类函数ntohl()把网络字节序转换成了长整形主机字节序。函数名中的n表示网络；h表示主机；l表示长整形；s表示短整形，即16位。

1.7　设定并获取默认的套接字超时时间

有时，你需要处理socket库某些属性的默认值，例如套接字超时时间。

1.7.1　实战演练

你可以创建一个套接字对象实例，调用gettimeout()方法获取默认的超时时间，调用settimeout()方法设定一个超时时间。这种操作在开发服务器应用时很有用。

在test_socket_timeout()函数中，首先创建一个套接字对象，然后使用读取或者设定实例方法处理超时时间。

代码清单1-6展示了如何定义test_socket_timeout()函数，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 1
# This program is optimized for Python 2.7. It may run on any
# other Python version with/without modifications.

import socket

def test_socket_timeout():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    print "Default socket timeout: %s" %s.gettimeout()
    s.settimeout(100)
    print "Current socket timeout: %s" %s.gettimeout()

if __name__ == '__main__':
    test_socket_timeout()
```

运行上述代码后，你会看到它是如何修改默认超时时间的，如下所示：

```
$ python 1_6_socket_timeout.py
Default socket timeout: None
Current socket timeout: 100.0
```

1.7.2 原理分析

在这段代码片段中，首先创建了一个套接字对象。套接字构造方法的第一个参数是地址族，第二个参数是套接字类型。然后，调用`gettimeout()`方法获取套接字超时时间，再调用`settimeout()`方法修改超时时间。传给`settimeout()`方法的参数可以是秒数（非负浮点数）也可以是`None`。这个方法在处理阻塞式套接字操作时使用。如果把超时时间设为`None`，则禁用了套接字操作的超时检测。

1.8 优雅地处理套接字错误

在网络应用中，经常会遇到这种情况：一方尝试连接，但另一方由于网络媒介失效或者其他原因无法响应。Python的`socket`库提供了一个方法，能通过`socket.error`异常优雅地处理套接字错误。在这个攻略中会举几个例子。

1.8.1 实战演练

我们来编写几个`try-except`代码块，每个块对应一种可能发生的错误。为了获取用户输入，可以使用`argparse`模块。这个模块的功能很强大，而不仅是可以使用`sys.argv`解析命令行参数。这些`try-except`代码块分别演示了常见的套接字操作，例如创建套接字对象、连接服务器、发送数据和等待应答。

下述攻略使用几行代码演示了如何处理异常。

代码清单1-7展示了如何处理`socket.error`异常，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 1
# This program is optimized for Python 2.7. It may run on any
# other Python version with/without modifications.

import sys
import socket
import argparse

def main():
    # setup argument parsing
    parser = argparse.ArgumentParser(description='Socket Error Examples')
    parser.add_argument('--host', action="store", dest="host", required=False)
    parser.add_argument('--port', action="store", dest="port", type=int, required=False)
```

```
    parser.add_argument('--file', action="store", dest="file", required=False)
    given_args = parser.parse_args()
    host = given_args.host
    port = given_args.port
    filename = given_args.file

    # First try-except block -- create socket
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    except socket.error, e:
        print "Error creating socket: %s" % e
        sys.exit(1)

    # Second try-except block -- connect to given host/port
    try:
        s.connect((host, port))
    except socket.gaierror, e:
        print "Address-related error connecting to server: %s" % e
        sys.exit(1)
    except socket.error, e:
        print "Connection error: %s" % e
        sys.exit(1)

    # Third try-except block -- sending data
    try:
        s.sendall("GET %s HTTP/1.0\r\n\r\n" % filename)
    except socket.error, e:
        print "Error sending data: %s" % e
        sys.exit(1)

    while 1:
        # Fourth tr-except block -- waiting to receive data from remote host
        try:
            buf = s.recv(2048)
        except socket.error, e:
            print "Error receiving data: %s" % e
            sys.exit(1)
        if not len(buf):
            break
        # write the received data
        sys.stdout.write(buf)

if __name__ == '__main__':
    main()
```

1.8.2　原理分析

在Python中，可以使用argparse模块把命令行参数传入脚本以及在脚本中解析命令行参数。这个模块在Python 2.7中可用。如果使用较旧版本的Python，这个模块可以到“Python包索引”（Python Package Index，简称PyPI）中获取，使用easy_install或pip安装。

这个攻略用到了三个命令行参数：主机名、端口号和文件名。上述脚本的使用方法如下：

```
$ python 1_7_socket_errors.py -host=<HOST> --port=<PORT> --file=<FILE>
```

如果提供的主机不存在，这个脚本会输出如下错误：

```
$ python 1_7_socket_errors.py --host=www.pytgo.org --port=8080
--file=1_7_socket_errors.py
Address-related error connecting to server: [Errno -5] No address associated with
hostname
```

如果某个端口上没有服务，你却尝试连接到这个端口，则这个脚本会抛出连接超时异常，如下所示：

```
$ python 1_7_socket_errors.py --host=www.python.org --port=8080
--file=1_7_socket_errors.py
```

这个命令会返回如下错误，因为主机www.python.org监听的不是端口8080：

```
Connection error: [Errno 110] Connection timed out
```

不过，如果向正确的主机、正确的端口发起随意的请求，应用层可能无法捕获这一异常。例如，运行下述脚本，不会返回错误，但输出的HTML代码说明了脚本的问题：

```
$ python 1_7_socket_errors.py --host=www.python.org --port=80
--file=1_7_socket_errors.py

HTTP/1.1 404 Not found
Server: Varnish
Retry-After: 0
content-type: text/html
Content-Length: 77
Accept-Ranges: bytes
Date: Thu, 20 Feb 2014 12:14:01 GMT
Via: 1.1 varnish
Age: 0
Connection: close

<html>
<head>
<title> </title>
</head>
<body>
unknown domain: </body></html>
```

这个攻略用到了四个try-except块。除第二个块处理socket.gaierror异常之外，其他块都处理socket.error异常。socket.gaierror是地址相关的错误。除此之外还有两种异常：socket.herror，C API中抛出的异常；如果在套接字中使用settimeout()方法，套接字超时后会抛出socket.timeout异常。

1.9　修改套接字发送和接收的缓冲区大小

很多情况下，默认的套接字缓冲区大小可能不够用。此时，可以将默认的套接字缓冲区大小改成一个更合适的值。

1.9.1　实战演练

我们要使用套接字对象的setsockopt()方法修改默认的套接字缓冲区大小。

首先，定义两个常量：SEND_BUF_SIZE和RECV_BUF_SIZE。然后在一个函数中调用套接字实例的setsockopt()方法。修改之前，最好先检查缓冲区大小是多少。注意，发送和接收的缓冲区大小要分开设定。

代码清单1-8展示了如何修改套接字的发送和接收缓冲区大小，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 1
# This program is optimized for Python 2.7. It may run on any
# other Python version with/without modifications.

import socket

SEND_BUF_SIZE = 4096
RECV_BUF_SIZE = 4096

def modify_buff_size():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM )

    # Get the size of the socket's send buffer
    bufsize = sock.getsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF)
    print "Buffer size [Before]:%d" %bufsize

    sock.setsockopt(socket.SOL_TCP, socket.TCP_NODELAY, 1)
    sock.setsockopt(
            socket.SOL_SOCKET,
            socket.SO_SNDBUF,
            SEND_BUF_SIZE)
    sock.setsockopt(
            socket.SOL_SOCKET,
            socket.SO_RCVBUF,
            RECV_BUF_SIZE)

    bufsize = sock.getsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF)
    print "Buffer size [After]:%d" %bufsize

if __name__ == '__main__':
    modify_buff_size()
```

运行上述脚本后，会显示修改套接字缓冲区大小前后的变化。根据你所用操作系统的本地设

定，得到的输出可能有所不同：

```
$ python 1_8_modify_buff_size.py
Buffer size [Before]:16384
Buffer size [After]:8192
```

1.9.2　原理分析

在套接字对象上可调用方法getsockopt()和setsockopt()分别获取和修改套接字对象的属性。setsockopt()方法接收三个参数：level、optname和value。其中，optname是选项名，value是该选项的值。第一个参数所用的符号常量（SO_*等）可在socket模块中查看。

1.10　把套接字改成阻塞或非阻塞模式

默认情况下，TCP套接字处于阻塞模式中。也就是说，除非完成了某项操作，否则不会把控制权交还给程序。例如，调用connect() API后，连接操作会阻止程序继续往下执行，直到连接成功为止。很多情况下，你并不想让程序一直等待服务器响应或者有异常终止操作。例如，如果编写了一个网页浏览器客户端连接服务器，你应该考虑提供取消功能，以便在操作过程中取消连接。这时就要把套接字设置为非阻塞模式。

1.10.1　实战演练

我们来看一下在Python中有哪些选项。在Python中，套接字可以被设置为阻塞模式或者非阻塞模式。在非阻塞模式中，调用API后，例如send()或recv()方法，如果遇到问题就会抛出异常。但在阻塞模式中，遇到错误并不会阻止操作。我们可以创建一个普通的TCP套接字，分别在阻塞模式和非阻塞模式中执行操作实验。

为了能在阻塞模式中处理套接字，首先要创建一个套接字对象。然后，调用setblocking(1)把套接字设为阻塞模式，或者调用setblocking(0)把套接字设为非阻塞模式。最后，把套接字绑定到指定的端口上，监听进入的连接。

代码清单1-9展示了如何把套接字设为阻塞模式或非阻塞模式，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 1
# This program is optimized for Python 2.7. It may run on any
# other Python version with/without modifications.

import socket
```

```
def test_socket_modes():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setblocking(1)
    s.settimeout(0.5)
    s.bind(("127.0.0.1", 0))

    socket_address = s.getsockname()
    print "Trivial Server launched on socket: %s" %str(socket_address)
    while(1):
        s.listen(1)

if __name__ == '__main__':
    test_socket_modes()
```

运行这个攻略后，会启动一个简易服务器，开启阻塞模式，如下述命令所示：

```
$ python 1_9_socket_modes.py
Trivial Server launched on socket: ('127.0.0.1', 51410)
```

1.10.2　原理分析

在这个攻略中，我们把1传给`setblocking()`方法，启用套接字的阻塞模式。类似地，可以把0传给这个方法，把套接字设为非阻塞模式。

这个功能在后面的一些攻略中会用到，到时再详细说明其真正作用。

1.11　重用套接字地址

不管连接是被有意还是无意关闭，有时你想始终在同一个端口上运行套接字服务器。某些情况下，如果客户端程序需要一直连接指定的服务器端口，这么做就很有用，因为无需改变服务器端口。

1.11.1　实战演练

如果在某个端口上运行一个Python套接字服务器，连接一次之后便终止运行，就不能再使用这个端口了。如果再次连接，程序会抛出如下错误：

```
Traceback (most recent call last):
  File "1_10_reuse_socket_address.py", line 40, in <module>
    reuse_socket_addr()
  File "1_10_reuse_socket_address.py", line 25, in reuse_socket_addr
    srv.bind( ('', local_port) )
  File "<string>", line 1, in bind
socket.error: [Errno 98] Address already in use
```

这个问题的解决方法是启用套接字重用选项`SO_REUSEADDR`。

创建套接字对象之后，我们可以查询地址重用的状态，比如说旧状态。然后，调用setsockopt()方法，修改地址重用状态的值。再按照常规的步骤，把套接字绑定到一个地址上，监听进入的客户端连接。在这个例子中，我们要捕获KeyboardInterrupt异常，这样按下Ctrl+C键后，Python脚本会终止运行，但不会显示任何异常消息。

代码清单1-10展示了如何重用套接字地址，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 1
# This program is optimized for Python 2.7. It may run on any
# other Python version with/without modifications.

import socket
import sys

def reuse_socket_addr():
    sock = socket.socket( socket.AF_INET, socket.SOCK_STREAM )

    # Get the old state of the SO_REUSEADDR option
    old_state = sock.getsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR)
    print "Old sock state: %s" %old_state

    # Enable the SO_REUSEADDR option
    sock.setsockopt( socket.SOL_SOCKET, socket.SO_REUSEADDR, 1 )
    new_state = sock.getsockopt( socket.SOL_SOCKET, socket.SO_REUSEADDR )
    print "New sock state: %s" %new_state

    local_port = 8282

    srv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    srv.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    srv.bind( ('', local_port) )
    srv.listen(1)
    print ("Listening on port: %s " %local_port)
    while True:
        try:
            connection, addr = srv.accept()
            print 'Connected by %s:%s' % (addr[0], addr[1])
        except KeyboardInterrupt:
            break
        except socket.error, msg:
            print '%s' % (msg,)

if __name__ == '__main__':
    reuse_socket_addr()
```

这个攻略的输出如下所示：

```
$ python 1_10_reuse_socket_address.py
Old sock state: 0
New sock state: 1
Listening on port: 8282
```

1.11.2　原理分析

你可以在一个终端窗口运行这个脚本，然后在另一个终端窗口中输入telnet localhost 8282，尝试连接这个服务器。关闭服务器程序后，还可以使用同一个端口再次连接。然而，如果你把设定SO_REUSEADDR的那行代码注释掉，服务器将不会再次运行脚本。

1.12　从网络时间服务器获取并打印当前时间

很多程序要求设备的时间精准，例如Unix系统中的make命令。设备上的时间可能不够准确，需要和网络中的时间服务器同步。

1.12.1　准备工作

你可以编写一个Python客户端，让设备上的时间和某个网络时间服务器同步。要完成这一操作，需要使用ntplib，通过“网络时间协议”（Network Time Protocol，简称NTP）处理客户端和服务器之间的通信。如果你的设备中没有安装ntplib，可以使用pip或easy_install从PyPI中安装，命令如下：

```
$pip install ntplib
```

1.12.2　实战演练

我们先要创建一个NTPClient实例，然后在这个实例上调用request()方法，把NTP服务器的地址传入方法。

代码清单1-11展示了如何从网络时间服务器上获取当前时间并打印出来，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 1
# This program is optimized for Python 2.7. It may run on any
# other Python version with/without modifications.

import ntplib
from time import ctime

def print_time():
    ntp_client = ntplib.NTPClient()
    response = ntp_client.request('pool.ntp.org')
    print ctime(response.tx_time)

if __name__ == '__main__':
    print_time()
```

在我的设备上，运行这个攻略后得到的输出如下：

```
$ python 1_11_print_machine_time.py
Thu Mar 5 14:02:58 2012
```

1.12.3 原理分析

在这个攻略中，我们编写了一个NTP客户端，向NTP服务器pool.ntp.org发起了一个NTP请求。响应使用ctime()函数打印出来。

1.13 编写一个 SNTP 客户端

与前一个攻略不同，有时并不需要从NTP服务器上获取精确的时间。遇到这种情况，就可以使用NTP的简化版本，叫作“简单网络时间协议”。

1.13.1 实战演练

让我们不使用任何第三方库编写一个简单的SNTP客户端。

首先，定义两个常量：NTP_SERVER和TIME1970。NTP_SERVER是客户端要连接的服务器地址，TIME1970指1970年1月1日（也叫Epoch）。在http://www.epochconverter.com/上可以查看Epoch时间值，或者把时间转换成Epoch时间值。这个客户端通过UDP协议创建一个UDP套接字（SOCK_DGRAM），用于连接服务器。然后，客户端要在一个数据包中把数据'\x1b' + 47 * '\0'发给SNTP服务器。UDP客户端分别使用sendto()和recvfrom()方法发送和接收数据。

服务器返回的时间信息打包在一个数组中，客户端需要使用struct模块取出数据。我们所需的数据是数组中的第11个元素。最后，我们要从取出的数据上减掉TIME1970，得到真正的当前时间。

代码清单1-12展示了如何编写这个SNTP客户端，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 1
# This program is optimized for Python 2.7. It may run on any
# other Python version with/without modifications.

import socket
import struct
import sys
import time

NTP_SERVER = "0.uk.pool.ntp.org"
```

```
TIME1970 = 2208988800L

def sntp_client():
    client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    data = '\x1b' + 47 * '\0'
    client.sendto(data, (NTP_SERVER, 123))
    data, address = client.recvfrom( 1024 )
    if data:
        print 'Response received from:', address
    t = struct.unpack( '!12I', data )[10]
    t -= TIME1970
    print '\tTime=%s' % time.ctime(t)

if __name__ == '__main__':
    sntp_client()
```

这个攻略通过SNTP协议从网络时间服务器上获取当前时间并打印出来，如下所示：

```
$ python 1_12_sntp_client.py
Response received from: ('87.117.251.2', 123)
      Time=Tue Feb 25 14:49:38 2014
```

1.13.2　原理分析

这个SNTP客户端创建一个套接字连接，然后通过协议发送数据。从NTP服务器（这里使用的是`0.uk.pool.ntp.org`）收到数据后，使用`struct`模块取出数据。最后，减去1970年1月1日对应的时间戳，再使用Python内置的`time`模块提供的`ctime()`方法打印时间。

1.14　编写一个简单的回显客户端/服务器应用

尝试过Python中`socket`模块的基本API后，现在我们来编写一个套接字服务器和客户端。这里，你将有机会利用在前述攻略中掌握的基本知识。

1.14.1　实战演练

在这个例子中，不管服务器从客户端收到什么输入，都会将其回显出来。我们要使用Python中的`argparse`模块，在命令行中指定TCP端口。服务器脚本和客户端脚本都要用到这个参数。

我们先来编写服务器。首先创建一个TCP套接字对象。然后设定启用重用地址，这样想运行多少次服务器就能运行多少次。我们把套接字绑定在本地设备的指定端口上。在监听阶段，把`backlog`参数传入`listen()`方法中，让服务器在队列中监听多个客户端。最后，等待客户端连接，向服务器发送一些数据。收到数据后，服务器会把数据回显给客户端。

代码清单1-13a展示了如何编写回显应用的服务器，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 1
# This program is optimized for Python 2.7. It may run on any
# other Python version with/without modifications.

import socket
import sys
import argparse

host = 'localhost'
data_payload = 2048
backlog = 5

def echo_server(port):
    """ A simple echo server """
    # Create a TCP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Enable reuse address/port
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    # Bind the socket to the port
    server_address = (host, port)
    print "Starting up echo server  on %s port %s" % server_address
    sock.bind(server_address)
    # Listen to clients, backlog argument specifies the max no. of queued connections
    sock.listen(backlog)
    while True:
        print "Waiting to receive message from client"
        client, address = sock.accept()
        data = client.recv(data_payload)
        if data:
            print "Data: %s" %data
            client.send(data)
            print "sent %s bytes back to %s" % (data, address)
        # end connection
        client.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int,
required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_server(port)
```

在客户端代码中，我们要创建一个客户端套接字，然后使用命令行参数中指定的端口连接服务器。客户端把消息`Test message. This will be echoed`发送给服务器之后，立即就会在几个数据片段中收到返回的消息。这里用到了两个`try-except`块，捕获交互过程中发生的任何异常。

代码清单1-13b展示了如何编写回显程序的客户端，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 1
# This program is optimized for Python 2.7. It may run on any
# other Python version with/without modifications.

import socket
import sys

import argparse

host = 'localhost'

def echo_client(port):
    """ A simple echo client """
    # Create a TCP/IP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Connect the socket to the server
    server_address = (host, port)
    print "Connecting to %s port %s" % server_address
    sock.connect(server_address)

    # Send data
    try:
        # Send data
        message = "Test message. This will be echoed"
        print "Sending %s" % message
        sock.sendall(message)
        # Look for the response
        amount_received = 0
        amount_expected = len(message)
        while amount_received < amount_expected:
            data = sock.recv(16)
            amount_received += len(data)
            print "Received: %s" % data
    except socket.errno, e:
        print "Socket error: %s" %str(e)
    except Exception, e:
        print "Other exception: %s" %str(e)
    finally:
        print "Closing connection to the server"
        sock.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int,
required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_client(port)
```

1.14.2 原理分析

为了查看客户端和服务器之间的交互，要在一个终端里启动如下服务器脚本：

```
$ python 1_13a_echo_server.py --port=9900
Starting up echo server  on localhost port 9900

Waiting to receive message from client
```

然后，在另一个终端里运行客户端，如下所示：

```
$ python 1_13b_echo_client.py --port=9900
Connecting to localhost port 9900
Sending Test message. This will be echoed
Received: Test message. Th
Received: is will be echoe
Received: d
Closing connection to the server
```

连接到本地主机后，服务器还会输出以下消息：

```
Data: Test message. This will be echoed
sent Test message. This will be echoed bytes back to ('127.0.0.1', 42961)
Waiting to receive message from client
```

第 2 章 使用多路复用套接字I/O提升性能

本章攻略：

- 在套接字服务器程序中使用`ForkingMixIn`
- 在套接字服务器程序中使用`ThreadingMixIn`
- 使用`select.select`编写一个聊天室服务器
- 使用`select.epoll`多路复用Web服务器
- 使用并发库Diesel多路复用回显服务器

2.1 简介

本章专注于使用一些有用的技术提升套接字服务器的性能。和前一章不同，本章考虑多个客户端连接服务器的情况，而且可以异步通信。服务器不需要在阻塞模式中处理客户端发出的请求，而是单独处理每个请求。如果某个客户端接收或处理数据时花了很长时间，服务器无需等待处理完成，可以使用另外的线程或进程和其他客户端通信。

本章还要介绍`select`模块。这个模块建立在底层操作系统内核的`select`系统调用基础之上，提供了平台专用的I/O监控功能。Linux用户可访问http://man7.org/linux/man-pages/man2/select.2.html查看手册，手册中介绍了`select`系统调用的可用功能。我们的套接字服务器要和多个客户端交互，所以`select`可以帮助我们监控非阻塞式套接字。有些第三方Python库也能帮助我们同时处理多个客户端，本章包含一个使用Diesel并发库的示例攻略。

为简单起见，我们只会使用少数几个客户端，但读者可以自行扩展本章的攻略，让它们处理几十甚至几百个客户端。

2.2 在套接字服务器程序中使用 ForkingMixIn

你已经决定要编写一个异步Python套接字服务器程序。服务器处理客户端发出的请求时不能阻塞，因此要找到一种机制来单独处理每个客户端。

Python 2.7版中的SocketServer模块提供了两个实用类：ForkingMixIn和ThreadingMixIn。ForkingMixIn会为每个客户端请求派生一个新进程。本节介绍ForkingMixIn类，ThreadingMixIn类将在下一节中介绍。有关SocketServer模块的详情，请参阅Python文档：http://docs.python.org/2/library/socketserver.html。

2.2.1 实战演练

我们要利用SocketServer模块提供的类，重写第1章中的回显服务器。SocketServer模块提供了可以直接使用的TCP、UDP及其他协议服务器。我们可以创建ForkingServer类，继承TCPServer和ForkingMixIn类。前一个父类让ForkingServer类实现了之前手动完成的所有服务器操作，例如创建套接字、绑定地址和监听进入的连接。我们的服务器还要继承ForkingMixIn类，异步处理客户端。

ForkingServer类还要创建一个请求处理程序，说明如何处理客户端请求。在这个攻略中，我们的服务器会回显客户端发送的文本字符串。请求处理类ForkingServerRequestHandler继承自SocketServer库提供的BaseRequestHandler类。

回显服务器的客户端ForkingClient可以使用面向对象的方式编写。在Python中，类的构造方法叫作__init__()。按照惯例，要把self作为参数传入__init__()方法，以便指定具体实例的属性。ForkingClient连接的回显服务器要在__init__()方法中初始化，然后在run()方法中向服务器发送消息。

如果你根本不知道“面向对象编程”（Object-oriented Programming，简称OOP），学习这个攻略之前最好熟悉一下OOP的基本概念。

若想测试ForkingServer类，可以启动多个回显客户端，看看服务器如何响应客户端。

代码清单2-1展示了如何在套接字服务器程序中使用ForkingMixIn类，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 2
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.
# See more: http://docs.python.org/2/library/socketserver.html

import os
import socket
```

```
import threading
import SocketServer

SERVER_HOST = 'localhost'
SERVER_PORT = 0 # tells the kernel to pick up a port dynamically
BUF_SIZE = 1024
ECHO_MSG = 'Hello echo server!'

class ForkingClient():
    """ A client to test forking server"""
    def __init__(self, ip, port):
        # Create a socket
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # Connect to the server
        self.sock.connect((ip, port))

    def run(self):
        """ Client playing with the server"""
        # Send the data to server
        current_process_id = os.getpid()
        print 'PID %s Sending echo message to the server : "%s"' % (current_process_id,
ECHO_MSG)
        sent_data_length = self.sock.send(ECHO_MSG)
        print "Sent: %d characters, so far..." %sent_data_length

        # Display server response
        response = self.sock.recv(BUF_SIZE)
        print "PID %s received: %s" % (current_process_id, response[5:])

    def shutdown(self):
        """ Cleanup the client socket """
        self.sock.close()

class ForkingServerRequestHandler(SocketServer.BaseRequestHandler):
    def handle(self):
        # Send the echo back to the client
        data = self.request.recv(BUF_SIZE)
        current_process_id = os.getpid()
        response = '%s: %s' % (current_process_id, data)
        print "Server sending response [current_process_id: data] = [%s]" %response
        self.request.send(response)
        return

class ForkingServer(SocketServer.ForkingMixIn,
                    SocketServer.TCPServer,
                    ):
    """Nothing to add here, inherited everything necessary from parents"""
    pass

def main():
    # Launch the server
    server = ForkingServer((SERVER_HOST, SERVER_PORT), ForkingServerRequestHandler)
    ip, port = server.server_address # Retrieve the port number
    server_thread = threading.Thread(target=server.serve_forever)
```

```
    server_thread.setDaemon(True) # don't hang on exit
    server_thread.start()
    print 'Server loop running PID: %s' %os.getpid()

    # Launch the client(s)
    client1 =  ForkingClient(ip, port)
    client1.run()

    client2 =  ForkingClient(ip, port)
    client2.run()

    # Clean them up
    server.shutdown()
    client1.shutdown()
    client2.shutdown()
    server.socket.close()

if __name__ == '__main__':
    main()
```


2.2.2　原理分析

主线程中创建了一个ForkingServer实例，作为守护进程在后台运行。然后再创建两个客户端和服务器交互。

运行这个脚本后，会看到如下输出：

```
$ python 2_1_forking_mixin_socket_server.py
Server loop running PID: 12608
PID 12608 Sending echo message to the server : "Hello echo server!"
Sent: 18 characters, so far...
Server sending response [current_process_id: data] = [12610: Hello echo server!]
PID 12608 received: : Hello echo server!
PID 12608 Sending echo message to the server : "Hello echo server!"
Sent: 18 characters, so far...
Server sending response [current_process_id: data] = [12611: Hello echo server!]
PID 12608 received: : Hello echo server!
```

在你的设备中可能会使用不同的服务器端口号，因为端口号由操作系统内核动态选择。

2.3　在套接字服务器程序中使用 ThreadingMixIn

或许基于某些原因你不想编写基于进程的应用程序，而更愿意编写多线程应用程序。可能的原因有：在线程之间共享应用的状态，避免进程间通信的复杂操作，等等。遇到这种需求，如果想使用SocketServer库编写异步网络服务器，就得使用ThreadingMixIn类。

2.3.1　准备工作

对前一个攻略做几处小改动就能使用ThreadingMixIn编写一个可用的套接字服务器。

> **下载示例代码**
>
> 如果你是通过http://www.packtpub.com的注册账户购买的图书，可以从该账户中下载相应Packt图书的示例代码。如果你是从其他地方购买的本书，可以访问http://www.packtpub.com/support，注册账户后，我们将会为你发送一封附有示例代码文件的电子邮件。

2.3.2　实战演练

和前一节中基于ForkingMixIn的套接字服务器一样，使用ThreadingMixIn编写的套接字服务器要遵循相同的回显服务器编程模式，不过仍有几点不同。首先，ThreadedTCPServer继承自TCPServer和TheadingMixIn。客户端连接这个多线程版服务器时，会创建一个新线程。详情参见http://docs.python.org/2/library/socketserver.html。

套接字服务器的请求处理类ForkingServerRequestHandler在一个新线程中把消息回显给客户端。在这个类中可以获取线程的信息。简单起见，我们把客户端的代码放在一个函数中，而不是一个类中。客户端代码创建客户端套接字，然后向服务器发送消息。

代码清单2-2展示了如何在回显套接字服务器中使用ThreadingMixIn类，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 2
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import os
import socket
import threading
import SocketServer

SERVER_HOST = 'localhost'
SERVER_PORT = 0 # tells the kernel to pickup a port dynamically
BUF_SIZE = 1024

def client(ip, port, message):
    """ A client to test threading mixin server"""
    # Connect to the server
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((ip, port))
    try:
        sock.sendall(message)
        response = sock.recv(BUF_SIZE)
```

```
        print "Client received: %s" %response
    finally:
        sock.close()

class ThreadedTCPRequestHandler(SocketServer.BaseRequestHandler):
    """ An example of threaded TCP request handler """
    def handle(self):
        data = self.request.recv(1024)
        current_thread = threading.current_thread()
        response = "%s: %s" %(current_thread.name, data)
        self.request.sendall(response)

class ThreadedTCPServer(SocketServer.ThreadingMixIn, SocketServer.TCPServer):
    """Nothing to add here, inherited everything necessary from parents"""
    pass

if __name__ == "__main__":
    # Run server
    server = ThreadedTCPServer((SERVER_HOST, SERVER_PORT),
ThreadedTCPRequestHandler)
    ip, port = server.server_address # retrieve ip address

    # Start a thread with the server -- one  thread per request
    server_thread = threading.Thread(target=server.serve_forever)
    # Exit the server thread when the main thread exits
    server_thread.daemon = True
    server_thread.start()
    print "Server loop running on thread: %s"  %server_thread.name

    # Run clients
    client(ip, port, "Hello from client 1")
    client(ip, port, "Hello from client 2")
    client(ip, port, "Hello from client 3")

    # Server cleanup
    server.shutdown()
```

2.3.3 原理分析

这个攻略首先创建一个服务器线程，并在后台启动。然后启动三个测试客户端，向服务器发送消息。作为响应，服务器把消息回显给客户端。在服务器请求处理类的`handle()`方法中，我们取回了当前线程的信息并将其打印出来，这些信息在每次客户端连接中都不同。

在客户端和服务器的通信中用到了`sendall()`方法，以保证发送的数据无任何丢失。

```
$ python 2_2_threading_mixin_socket_server.py
Server loop running on thread: Thread-1
Client received: Thread-2: Hello from client 1
Client received: Thread-3: Hello from client 2
Client received: Thread-4: Hello from client 3
```

2.4　使用 select.select 编写一个聊天室服务器

在大型网络服务器应用程序中可能有几百或几千个客户端同时连接服务器，此时为每个客户端创建单独的线程或进程可能不切实际。由于内存可用量受限，且主机的CPU能力有限，我们需要一种更好的技术来处理大量的客户端。幸好，Python提供的select模块能解决这一问题。

2.4.1　实战演练

我们将编写一个高效的聊天室服务器，处理几百或更多数量的客户端连接。我们要使用select模块提供的select()方法，让聊天室服务器和客户端所做的操作始终不会阻塞消息的发送和接收。

这个攻略使用一个脚本就能启动客户端和服务器，执行脚本时要指定--name参数。只有在命令行中传入了--name=server，脚本才启动聊天室服务器。如果为--name参数指定了其他值，例如client1或client2，则脚本会启动聊天室客户端。聊天室服务器绑定的端口在命令行参数--port中指定。对大型应用程序而言，最好在不同的模块中编写服务器和客户端。

代码清单2-3展示了一个使用select.select编写的聊天室应用示例，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 2
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import select
import socket
import sys
import signal
import cPickle
import struct
import argparse

SERVER_HOST = 'localhost'
CHAT_SERVER_NAME = 'server'

# Some utilities
def send(channel, *args):
    buffer = cPickle.dumps(args)
    value = socket.htonl(len(buffer))
    size = struct.pack("L",value)
    channel.send(size)
    channel.send(buffer)

def receive(channel):
    size = struct.calcsize("L")
    size = channel.recv(size)
    try:
```

```
        size = socket.ntohl(struct.unpack("L", size)[0])
    except struct.error, e:
        return ''
    buf = ""
    while len(buf) < size:
        buf = channel.recv(size - len(buf))
    return cPickle.loads(buf)[0]
```


send()函数接收一个具名参数channel和一个定位参数*args，使用cPickle模块中的dumps()方法序列化数据，使用struct模块计算数据的大小。同样，receive()函数也接收一个具名参数channel。

然后定义ChatServer类，如下所示：

```
class ChatServer(object):
    """ An example chat server using select """
    def __init__(self, port, backlog=5):
        self.clients = 0
        self.clientmap = {}
        self.outputs = [] # list output sockets
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.server.bind((SERVER_HOST, port))
        print 'Server listening to port: %s ...' %port
        self.server.listen(backlog)
        # Catch keyboard interrupts
        signal.signal(signal.SIGINT, self.sighandler)

    def sighandler(self, signum, frame):
        """ Clean up client outputs"""
        # Close the server
        print 'Shutting down server...'
        # Close existing client sockets
        for output in self.outputs:
            output.close()
        self.server.close()

    def get_client_name(self, client):
        """ Return the name of the client """
        info = self.clientmap[client]
        host, name = info[0][0], info[1]
        return '@'.join((name, host))
```

ChatServer类的主要执行方法如下所示：

```
def run(self):
    inputs = [self.server, sys.stdin]
    self.outputs = []
    running = True
    while running:
        try:
            readable, writeable, exceptional = select.select(inputs, self.outputs, [])
```

```
        except select.error, e:
            break

        for sock in readable:
            if sock == self.server:
                # handle the server socket
                client, address = self.server.accept()
                print "Chat server: got connection %d from %s" % (client.fileno(),
address)
                # Read the login name
                cname = receive(client).split('NAME: ')[1]

                # Compute client name and send back
                self.clients += 1
                send(client, 'CLIENT: ' + str(address[0]))
                inputs.append(client)
                self.clientmap[client] = (address, cname)
                # Send joining information to other clients
                msg = "\n(Connected: New client (%d) from %s)" % (self.clients,
self.get_client_name(client))
                for output in self.outputs:
                    send(output, msg)
                self.outputs.append(client)

            elif sock == sys.stdin:
                # handle standard input
                junk = sys.stdin.readline()
                running = False
            else:
                # handle all other sockets
                try:
                    data = receive(sock)
                    if data:
                        # Send as new client's message...
                        msg = '\n#[' + self.get_client_name(sock) + ']>>' + data
                        # Send data to all except ourself
                        for output in self.outputs:
                            if output != sock:
                                send(output, msg)
                    else:
                        print "Chat server: %d hung up" % sock.fileno()
                        self.clients -= 1
                        sock.close()
                        inputs.remove(sock)
                        self.outputs.remove(sock)

                        # Sending client leaving information to others
                        msg = "\n(Now hung up: Client from %s)" % self.get_client_
name(sock)
                        for output in self.outputs:
                            send(output, msg)
                except socket.error, e:
                    # Remove
                    inputs.remove(sock)
```

```
                self.outputs.remove(sock)
        self.server.close()
```

初始化聊天室服务器时创建了一些属性：客户端数量、客户端映射和输出的套接字。和之前创建服务器套接字一样，初始化时也设定了重用地址的选项，这么做可以使用同一个端口重启服务器。聊天室服务器类的构造方法还有一个可选参数backlog，用于设定服务器监听的连接队列的最大数量。

这个聊天室服务器有个值得介绍的地方，它可以使用signal模块捕获用户的中断操作。中断操作一般通过键盘输入。ChatServer类为中断信号（SIGINT）注册了一个信号处理方法sighandler。信号处理方法捕获从键盘输入的中断信号后，关闭所有输出套接字，其中一些套接字可能还有数据等待发送。

聊天室服务器的主要执行方法是run()，在while循环中执行操作。run()方法注册了一个select接口，输入参数是聊天室服务器套接字stdin，输出参数由服务器的输出套接字列表指定。调用select.select()方法后得到三个列表：可读套接字、可写套接字和异常套接字。聊天室服务器只关心可读套接字，其中保存了准备被读取的数据。如果可读套接字是服务器本身，表示有一个新客户端连到服务器上了，服务器会读取客户端的名字，将其广播给其他客户端。如果输入参数中有内容，聊天室服务器会退出。类似地，这个聊天室服务器也能处理其他客户端套接字的输入，转播客户端直接传送的数据，还能共享客户端进入和离开聊天室的信息。

聊天室客户端应该包含以下代码：

```
class ChatClient(object):
    """ A command line chat client using select """

    def __init__(self, name, port, host=SERVER_HOST):
        self.name = name
        self.connected = False
        self.host = host
        self.port = port
        # Initial prompt
        self.prompt='[' + '@'.join((name, socket.gethostname().split('.')[0])) + ']> '
        # Connect to server at port
        try:
            self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.sock.connect((host, self.port))
            print "Now connected to chat server@ port %d" % self.port
            self.connected = True
            # Send my name...
            send(self.sock,'NAME: ' + self.name)
            data = receive(self.sock)
            # Contains client address, set it
            addr = data.split('CLIENT: ')[1]
            self.prompt = '[' + '@'.join((self.name, addr)) + ']> '
        except socket.error, e:
            print "Failed to connect to chat server @ port %d" % self.port
```

```
                sys.exit(1)

    def run(self):
        """ Chat client main loop """
        while self.connected:
            try:
                sys.stdout.write(self.prompt)
                sys.stdout.flush()
                # Wait for input from stdin and socket
                readable, writeable,exceptional = select.select([0, self.sock], [],[])
                for sock in readable:
                    if sock == 0:
                        data = sys.stdin.readline().strip()
                        if data: send(self.sock, data)
                    elif sock == self.sock:
                        data = receive(self.sock)
                        if not data:
                            print 'Client shutting down.'
                            self.connected = False
                            break
                        else:
                            sys.stdout.write(data + '\n')
                            sys.stdout.flush()

            except KeyboardInterrupt:
                print " Client interrupted. """
                self.sock.close()
                break
```

初始化聊天室客户端时指定了name参数，连接到聊天室服务器之后，这个名字会发送给服务器。初始化时还设置了一个自定义的提示符[name@host]>。客户端的执行方法run()在连接到服务器的过程中一直运行着。和聊天室服务器类似，聊天室客户端也使用select()方法注册。只要可读套接字做好了准备，客户端就开始接收数据。如果sock的值为0，而且有可用的数据，客户端就可以发送数据。发送的数据还会显示在stdout或者本例中的命令行终端里。主方法应该接收命令行参数，调用服务器或者客户端，如下所示：

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Socket Server Example with Select')
    parser.add_argument('--name', action="store", dest="name", required=True)
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    name = given_args.name
    if name == CHAT_SERVER_NAME:
        server = ChatServer(port)
        server.run()
    else:
        client = ChatClient(name=name, port=port)
        client.run()
```

这个脚本要运行三次：一次用于启动聊天室服务器，两次用于启动两个聊天室客户端。启动

服务器时，在命令行中传入参数--name=server和--port=8800。启动client1时，把名字参数改成--name=client1；启动client2时改为--name=client2。然后在client1中发送消息"Hello from client 1"，这个消息会显示在client2的终端里。同样，在client2中发送消息"hello from client 2"，也会在client1的终端里显示。

服务器的输出如下：

```
$ python 2_3_chat_server_with_select.py --name=server --port=8800
Server listening to port: 8800 ...
Chat server: got connection 4 from ('127.0.0.1', 56565)
Chat server: got connection 5 from ('127.0.0.1', 56566)
```

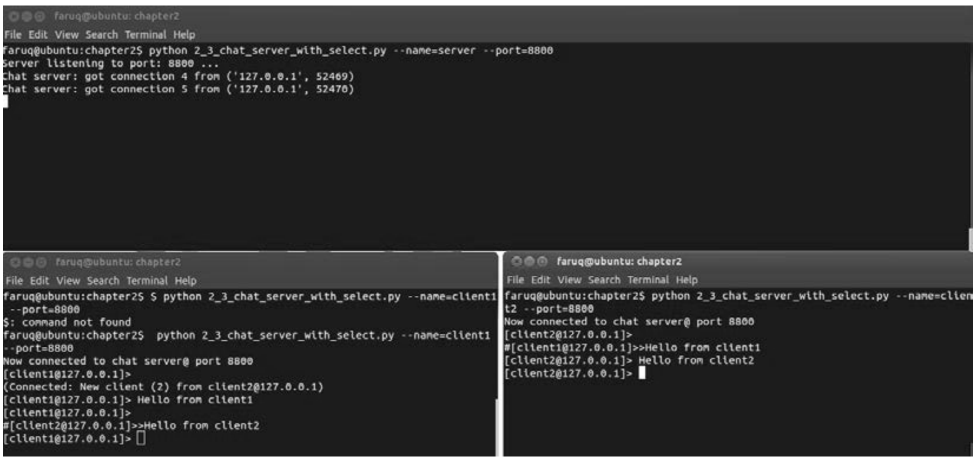
client1的输出如下：

```
$ python 2_3_chat_server_with_select.py --name=client1 --port=8800
Now connected to chat server@ port 8800
[client1@127.0.0.1]>
(Connected: New client (2) from client2@127.0.0.1)
[client1@127.0.0.1]> Hello from client 1
[client1@127.0.0.1]>
#[client2@127.0.0.1]>>hello from client 2
```

client2的输出如下：

```
$ python 2_3_chat_server_with_select.py --name=client2 --port=8800
Now connected to chat server@ port 8800
[client2@127.0.0.1]>
#[client1@127.0.0.1]>>Hello from client 1
[client2@127.0.0.1]> hello from client 2
[client2@127.0.0.1]
```

整个交互过程如下面的截图所示：



2.4.2　原理分析

在这个模块的顶端定义了两个实用函数：send()和receive()。

在聊天室服务器和客户端中用到了这两个函数，如前面的代码所示。聊天室服务器和客户端中定义的方法前面也介绍过了。

2.5　使用 select.epoll 多路复用 Web 服务器

Python的select模块中有很多针对特定平台的网络事件管理函数。在Linux设备中可以使用epoll。这个函数利用操作系统内核轮询网络事件，让脚本知道有事件发生了。这听起来比前面介绍的select.select方案更高效。

2.5.1　实战演练

我们来编写一个简单的Web服务器，向每一个连接服务器的网页浏览器返回一行文本。

这个脚本的核心在Web服务器的初始化过程中，我们要调用方法select.epoll()，注册服务器的文件描述符，以达到事件通知的目的。在Web服务器执行的代码中，套接字事件由下述代码监控。

代码清单2-4展示了如何使用select.epoll实现简单的Web服务器，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 2
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import socket
import select
import argparse

SERVER_HOST = 'localhost'

EOL1 = b'\n\n'
EOL2 = b'\n\r\n'
SERVER_RESPONSE  = b"""HTTP/1.1 200 OK\r\nDate: Mon, 1 Apr 2013 01:01:01
GMT\r\nContent-Type: text/plain\r\nContent-Length: 25\r\n\r\n
Hello from Epoll Server!"""

class EpollServer(object):
    """ A socket server using Epoll"""

    def __init__(self, host=SERVER_HOST, port=0):
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

```
        self.sock.bind((host, port))
        self.sock.listen(1)
        self.sock.setblocking(0)
        self.sock.setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)
        print "Started Epoll Server"
        self.epoll = select.epoll()
        self.epoll.register(self.sock.fileno(), select.EPOLLIN)

    def run(self):
        """Executes epoll server operation"""
        try:
            connections = {}; requests = {}; responses = {}
            while True:
                events = self.epoll.poll(1)
                for fileno, event in events:
                    if fileno == self.sock.fileno():
                        connection, address = self.sock.accept()
                        connection.setblocking(0)
                        self.epoll.register(connection.fileno(), select.EPOLLIN)
                        connections[connection.fileno()] = connection
                        requests[connection.fileno()] = b''
                        responses[connection.fileno()] = SERVER_RESPONSE
                    elif event & select.EPOLLIN:
                        requests[fileno] += connections[fileno].recv(1024)
                        if EOL1 in requests[fileno] or EOL2 in requests[fileno]:
                            self.epoll.modify(fileno, select.EPOLLOUT)
                            print('-'*40 + '\n' + requests[fileno].decode()[:-2])
                    elif event & select.EPOLLOUT:
                        byteswritten = connections[fileno].send(responses[fileno])
                        responses[fileno] = responses[fileno][byteswritten:]
                        if len(responses[fileno]) == 0:
                            self.epoll.modify(fileno, 0)
                            connections[fileno].shutdown(socket.SHUT_RDWR)
                    elif event & select.EPOLLHUP:
                        self.epoll.unregister(fileno)
                        connections[fileno].close()
                        del connections[fileno]
        finally:
            self.epoll.unregister(self.sock.fileno())
            self.epoll.close()
            self.sock.close()


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example with Epoll')
    parser.add_argument('--port', action="store", dest="port", type=int,
required=True)
    given_args = parser.parse_args()
    port = given_args.port
    server = EpollServer(host=SERVER_HOST, port=port)
    server.run()
```

运行这个脚本，在网页浏览器（例如Firefox或IE）中输入http://localhost:8800/访问服务器，

在终端会看到如下输出：

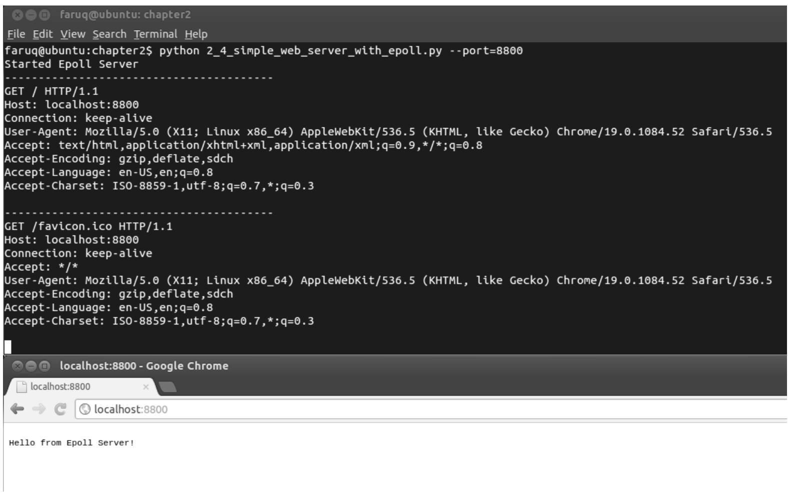
```
$ python 2_4_simple_web_server_with_epoll.py --port=8800
Started Epoll Server
----------------------------------------
GET / HTTP/1.1
Host: localhost:8800
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.31 (KHTML, like Gecko)
Chrome/26.0.1410.43 Safari/537.31
DNT: 1
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Cookie: MoodleSession=69149dqnvhett7br3qebsrcmh1;MOODLEID1_=%257F%25BA%2B%2540V

----------------------------------------
GET /favicon.ico HTTP/1.1
Host: localhost:8800
Connection: keep-alive
Accept: */*
DNT: 1
User-Agent: Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.31 (KHTML, like Gecko)
Chrome/26.0.1410.43 Safari/537.31
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

在浏览器中还会看到以下这行文本：

```
Hello from Epoll Server!
```

这一过程的截图如下所示：



2.5.2　原理分析

在Web服务器EpollServer的构造方法中创建了一个套接字服务器，绑定到本地主机的指定端口上。服务器的套接字被设定为非阻塞模式（setblocking(0)），并设定了TCP_NODELAY选项，让服务器无需缓冲便可直接交换数据（比如在SSH连接中）。然后创建了一个select.epoll()实例，再把套接字的文件描述符传给这个实例，以便监控。

在这个Web服务器的run()方法中开始监听套接字事件。事件由下述常量表示：

- EPOLLIN：套接字读事件
- EPOLLOUT：套接字写事件

这个套接字服务器把响应设为SERVER_RESPONSE。如果连接套接字服务器的客户端想写数据，可以在EPOLLOUT事件中完成。发生内部错误时，EPOLLHUP事件会把一个异常关闭信号发给套接字服务器。

2.6　使用并发库 Diesel 多路复用回显服务器

有时你需要编写一个大型自定义网络应用程序，但不想重复输入初始化服务器的代码，比如说创建套接字、绑定地址、监听以及处理基本的错误等。有很多Python网络库都可以帮助你把样板代码删除。这里我们要使用一个提供这种功能的库，它叫作Diesel。

2.6.1　准备工作

Diesel使用非阻塞和协程技术提升编写网络服务器的效率。Diesel的网站上有这么一句话：“Diesel的核心是一个紧密的事件轮询，使用epoll提供几近平稳的性能，即便有10 000个或更多的连接也无妨。”这一节我们通过一个简单的回显服务器介绍Diesel的用法。你需要安装Diesel 3.0或者更新的版本，使用pip命令即可完成：$ pip install diesel >= 3.0。

2.6.2　实战演练

在Python的Diesel框架中，应用程序使用Application()类的实例初始化，事件处理函数注册在这个实例上。我们来看一下使用Diesel编写回显服务器是多么简单。

代码清单2-5展示了如何使用Diesel编写回显服务器，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 2
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.
```

```
# You alos need diesel library 3.0 or any later version

import diesel
import argparse

class EchoServer(object):
    """ An echo server using diesel"""

    def handler(self, remote_addr):
        """Runs the echo server"""
        host, port = remote_addr[0], remote_addr[1]
        print "Echo client connected from: %s:%d" %(host, port)

        while True:
            try:
                message = diesel.until_eol()
                your_message = ': '.join(['You said', message])
                diesel.send(your_message)
            except Exception, e:
                print "Exception:",e

def main(server_port):
    app = diesel.Application()
    server = EchoServer()
    app.add_service(diesel.Service(server.handler, server_port))
    app.run()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Echo server example with Diesel')
    parser.add_argument('--port', action="store", dest="port", type=int,
required=True)
    given_args = parser.parse_args()
    port = given_args.port
    main(port)
```

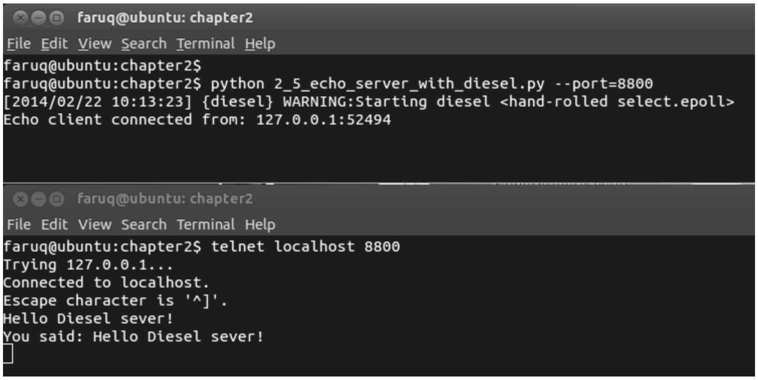
运行这个脚本后，服务器会显示如下输出：

```
$ python 2_5_echo_server_with_diesel.py --port=8800
[2013/04/08 11:48:32] {diesel} WARNING:Starting diesel <hand-rolled select.epoll>
Echo client connected from: 127.0.0.1:56603
```

在另一个终端窗口中可以使用Telnet客户端连接回显服务器，测试消息回显，如下所示：

```
$ telnet localhost 8800
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Hello Diesel server?
You said: Hello Diesel server?
```

下面这个截图显示了和Diesel聊天室服务器交互的过程：



2.6.3 原理分析

这个脚本从命令行参数`--port`中获取端口号，将其传给`main()`函数。Diesel应用程序在`main()`函数中初始化并运行。

在Diesel中有“服务”的概念，应用程序可以提供多种服务。`EchoServer`类中定义了`handler()`方法，让服务器能够处理单独的客户端连接。运行服务时要把`handler()`方法和端口号作为参数传给`Service()`方法。

`handler()`方法决定服务器的行为，在这个脚本中，服务器直接返回消息文本。

如果把这个脚本和第1章中的“编写一个简单的回显客户端/服务器应用”攻略（代码清单1-13a）对比，很明显能看出，我们不需要编写样板代码，因此很容易把精力集中在高层应用逻辑上。

第 3 章

IPv6、Unix域套接字和网络接口

本章攻略：

- 把本地端口转发到远程主机
- 通过ICMP查验网络中的主机
- 等待远程网络服务上线
- 枚举设备中的接口
- 找出设备中某个接口的IP地址
- 探测设备中的接口是否开启
- 检测网络中未开启的设备
- 使用相连的套接字执行基本的进程间通信
- 使用Unix域套接字执行进程间通信
- 确认你使用的Python是否支持IPv6套接字
- 从IPv6地址中提取IPv6前缀
- 编写一个IPv6回显客户端/服务器

3.1 简介

本章使用一些第三方库扩展Python中`socket`库的用法，还要介绍一些高级技术，例如Python标准库中的`ayncore`异步模块。与此同时，还会涉及很多不同的协议，例如ICMP查验和IPv6客户端/服务器。

本章通过一些示例攻略介绍几个有用的Python第三方模块的用法，例如Python网络程序员熟知的网络数据包抓取库Scapy。

本章部分攻略专门介绍Python中IPv6的处理方法，包括开发一个IPv6客户端/服务器应用程序。其他攻略则涉及Unix域套接字。

3.2 把本地端口转发到远程主机

有时，你需要创建一个本地端口转发器，把本地端口发送的流量全部重定向到特定的远程主机上。利用这个功能，可以让用户只能访问特定的网站，而不能访问其他网站。

3.2.1 实战演练

我们来编写一个本地端口转发脚本，把8800端口接收到的所有流量重定向到谷歌的首页（http://www.google.com）。我们可以把本地主机和远程主机连同端口号一起传入脚本。简单起见，这里只指定本地端口号，因为我们知道Web服务器运行在80端口上。

代码清单3-1是一个端口转发示例，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 3
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse

LOCAL_SERVER_HOST = 'localhost'
REMOTE_SERVER_HOST = 'www.google.com'
BUFSIZE = 4096

import asyncore
import socket
```

首先，我们来定义`PortForwarder`类：

```
class PortForwarder(asyncore.dispatcher):
    def __init__(self, ip, port, remoteip,remoteport,backlog=5):
        asyncore.dispatcher.__init__(self)
        self.remoteip=remoteip
        self.remoteport=remoteport
        self.create_socket(socket.AF_INET,socket.SOCK_STREAM)
        self.set_reuse_addr()
        self.bind((ip,port))
        self.listen(backlog)

    def handle_accept(self):
        conn, addr = self.accept()
        print "Connected to:",addr
        Sender(Receiver(conn),self.remoteip,self.remoteport)
```

然后定义`Receiver`和`Sender`类，如下所示：

```
class Receiver(asyncore.dispatcher):
    def __init__(self,conn):
        asyncore.dispatcher.__init__(self,conn)
```

```
        self.from_remote_buffer=''
        self.to_remote_buffer=''
        self.sender=None

    def handle_connect(self):
        pass

    def handle_read(self):
        read = self.recv(BUFSIZE)
        self.from_remote_buffer += read

    def writable(self):
        return (len(self.to_remote_buffer) > 0)

    def handle_write(self):
        sent = self.send(self.to_remote_buffer)
        self.to_remote_buffer = self.to_remote_buffer[sent:]

    def handle_close(self):
        self.close()
        if self.sender:
            self.sender.close()

class Sender(asyncore.dispatcher):
    def __init__(self, receiver, remoteaddr,remoteport):
        asyncore.dispatcher.__init__(self)
        self.receiver=receiver
        receiver.sender=self
        self.create_socket(socket.AF_INET, socket.SOCK_STREAM)
        self.connect((remoteaddr, remoteport))

    def handle_connect(self):
        pass

    def handle_read(self):
        read = self.recv(BUFSIZE)
        self.receiver.to_remote_buffer += read

    def writable(self):
        return (len(self.receiver.from_remote_buffer) > 0)

    def handle_write(self):
        sent = self.send(self.receiver.from_remote_buffer)
        self.receiver.from_remote_buffer = self.receiver.from_remote_buffer[sent:]

    def handle_close(self):
        self.close()
        self.receiver.close()

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Stackless Socket Server Example')
    parser.add_argument('--local-host', action="store", dest="local_host",
default=LOCAL_SERVER_HOST)
    parser.add_argument('--local-port', action="store", dest="local_port", type=int,
```

```
required=True)
    parser.add_argument('--remote-host', action="store", dest="remote_host",
default=REMOTE_SERVER_HOST)
    parser.add_argument('--remote-port', action="store", dest="remote_port",
type=int, default=80)
    given_args = parser.parse_args()
    local_host, remote_host = given_args.local_host, given_args.remote_host
    local_port, remote_port = given_args.local_port, given_args.remote_port

    print "Starting port forwarding local %s:%s => remote %s:%s" % (local_host,
local_port, remote_host, remote_port)
    PortForwarder(local_host, local_port, remote_host, remote_port)
    asyncore.loop()
```

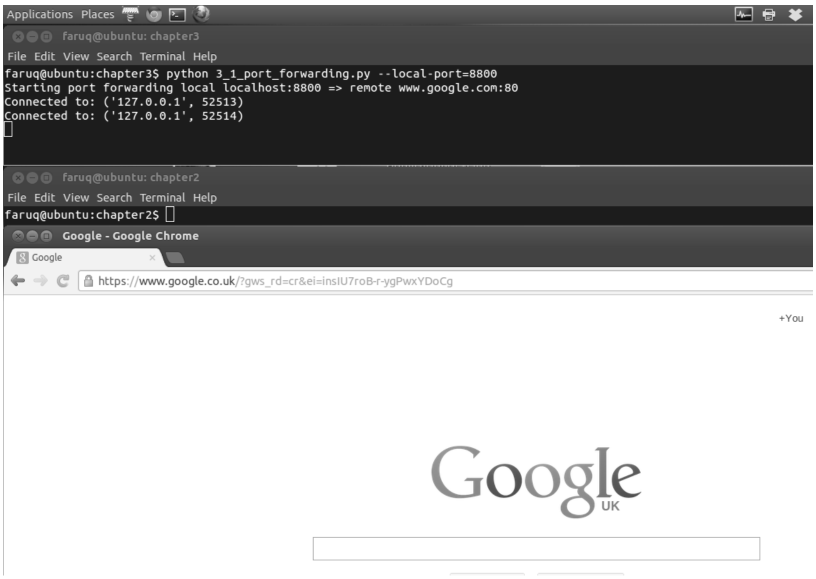

运行这个脚本后，会看到如下输出：

```
$ python 3_1_port_forwarding.py --local-port=8800
Starting port forwarding local localhost:8800 => remote www.google.com:80
```

现在打开浏览器，访问http://localhost:8800。浏览器会把你带到谷歌的首页，在命令行中会输出类似下面的信息：

```
Connected to: ('127.0.0.1', 38557)
```

把本地端口转发到远程主机的过程如下面的截图所示：



3.2.2　原理分析

我们创建了一个端口转发类`PortForwarder`，继承自`asyncore.dispatcher`。`asyncore.`

dispatcher类包装了一个套接字对象，还提供了一些帮助方法用于处理特定的事件，例如连接成功或客户端连接到服务器套接字。你可以选择重定义这些方法，在上面的脚本中我们只重定义了handle_accept()方法。

另外两个类也继承自asyncore.dispatcher。Receiver类处理进入的客户端请求，Sender类接收一个Receiver类实例，把数据发送给客户端。如你所见，这两个类都重定义了handle_read()、handle_write()和writeable()三个方法，目的是实现远程主机和本地客户端之间的双向通信。

概括来说，PortForwarder类在一个本地套接字中保存进入的客户端请求，然后把这个套接字传给Sender类实例，再使用Receiver类实例发起与远程主机指定端口之间的双向通信。

3.3　通过 ICMP 查验网络中的主机

ICMP查验（ICMP ping）[①]是你见过的最普通的网络扫描类型。ICMP查验做起来很简单，打开命令行或终端，输入ping www.google.com即可。这在Python程序中又有什么难的呢？这个攻略展示了一个简单的Python查验脚本。

3.3.1　准备工作

要在你的设备上运行这个脚本，需要有超级用户或管理员权限才行。

3.3.2　实战演练

你可以偷个懒，在Python脚本中调用系统中的ping命令，如下所示：

```
import subprocess
import shlex

command_line = "ping -c 1 www.google.com"
args = shlex.split(command_line)
try:
    subprocess.check_call(args,stdout=subprocess.PIPE,\
        stderr=subprocess.PIPE)
    print "Google web server is up!"
except subprocess.CalledProcessError:
    print "Failed to get ping."
```

然而，很多情况下，系统中的ping可执行文件不可用，或者无法访问。此时，我们需要一个纯粹的Python脚本实现查验。注意，这个脚本要使用超级用户或者管理员的身份运行。

① ICMP是Internet Control Message Protocol的简称，意思是“网络控制报文协议”。——译者注

代码清单3-2展示了如何执行ICMP查验，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 3
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import os
import argparse
import socket
import struct
import select
import time

ICMP_ECHO_REQUEST = 8 # Platform specific
DEFAULT_TIMEOUT = 2
DEFAULT_COUNT = 4

class Pinger(object):
    """ Pings to a host -- the Pythonic way"""

    def __init__(self, target_host, count=DEFAULT_COUNT, timeout=DEFAULT_TIMEOUT):
        self.target_host = target_host
        self.count = count
        self.timeout = timeout

    def do_checksum(self, source_string):
        """  Verify the packet integritity """
        sum = 0
        max_count = (len(source_string)/2)*2
        count = 0
        while count < max_count:
            val = ord(source_string[count + 1])*256 + ord(source_string[count])
            sum = sum + val
            sum = sum & 0xffffffff
            count = count + 2

        if max_count<len(source_string):
            sum = sum + ord(source_string[len(source_string) - 1])
            sum = sum & 0xffffffff

        sum = (sum >> 16)  +  (sum & 0xffff)
        sum = sum + (sum >> 16)
        answer = ~sum
        answer = answer & 0xffff
        answer = answer >> 8 | (answer << 8 & 0xff00)
        return answer

    def receive_pong(self, sock, ID, timeout):
        """
        Receive ping from the socket.
        """
        time_remaining = timeout
```

3

```
        while True:
            start_time = time.time()
            readable = select.select([sock], [], [], time_remaining)
            time_spent = (time.time() - start_time)
            if readable[0] == []: # Timeout
                return

            time_received = time.time()
            recv_packet, addr = sock.recvfrom(1024)
            icmp_header = recv_packet[20:28]
            type, code, checksum, packet_ID, sequence = struct.unpack(
                "bbHHh", icmp_header
            )
            if packet_ID == ID:
                bytes_In_double = struct.calcsize("d")
                time_sent = struct.unpack("d", recv_packet[28:28 + bytes_In_double])[0]
                return time_received - time_sent

            time_remaining = time_remaining - time_spent
            if time_remaining <= 0:
                return
```

我们要定义send_ping()方法，把查验请求的数据发送给目标主机。而且，在这个方法中还要调用do_checksum()方法，检查查验数据的完整性，如下所示：

```
    def send_ping(self, sock,  ID):
        """
        Send ping to the target host
        """
        target_addr  =  socket.gethostbyname(self.target_host)

        my_checksum = 0

        # Create a dummy heder with a 0 checksum.
        header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, my_checksum, ID, 1)
        bytes_In_double = struct.calcsize("d")
        data = (192 - bytes_In_double) * "Q"
        data = struct.pack("d", time.time()) + data

        # Get the checksum on the data and the dummy header.
        my_checksum = self.do_checksum(header + data)
        header = struct.pack(
            "bbHHh", ICMP_ECHO_REQUEST, 0, socket.htons(my_checksum), ID, 1
        )
        packet = header + data
        sock.sendto(packet, (target_addr, 1))
```

我们再来定义一个方法，ping_once()，只向目标主机发送一次查验。在这个方法中，把ICMP协议传给socket()方法，创建一个原始的ICMP套接字。异常处理代码负责处理未使用超级用户运行脚本的情况，以及其他套接字错误。代码如下：

```
    def ping_once(self):
        """
        Returns the delay (in seconds) or none on timeout.
        """
        icmp = socket.getprotobyname("icmp")
        try:
            sock = socket.socket(socket.AF_INET, socket.SOCK_RAW, icmp)
        except socket.error, (errno, msg):
            if errno == 1:
                # Not superuser, so operation not permitted
                msg +=  "ICMP messages can only be sent from root user processes"
                raise socket.error(msg)
        except Exception, e:
            print "Exception: %s" %(e)

        my_ID = os.getpid() & 0xFFFF

        self.send_ping(sock, my_ID)
        delay = self.receive_pong(sock, my_ID, self.timeout)
        sock.close()
        return delay
```

这个类要执行的主方法是ping()。这个方法中有个for循环，在for循环中调用ping_once()方法count次。延迟时间从查验的响应中获取，单位为秒。如果没有返回延迟时间，就意味着查验失败。代码如下：

```
        def ping(self):
            """
            Run the ping process
            """
            for i in xrange(self.count):
                print "Ping to %s..." % self.target_host,
                try:
                    delay  =  self.ping_once()
                except socket.gaierror, e:
                    print "Ping failed. (socket error: '%s')" % e[1]
                    break

                if delay  ==  None:
                    print "Ping failed. (timeout within %ssec.)" % self.timeout
                else:
                    delay  =  delay * 1000
                    print "Get pong in %0.4fms" % delay

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Python ping')
    parser.add_argument('--target-host', action="store", dest="target_host",
required=True)
    given_args = parser.parse_args()
    target_host = given_args.target_host
    pinger = Pinger(target_host=target_host)
    pinger.ping()
```

以超级用户的身份运行这个脚本，得到的输出如下所示：

```
$ sudo python 3_2_ping_remote_host.py --target-host=www.google.com
Ping to www.google.com... Get pong in 7.6921ms
Ping to www.google.com... Get pong in 7.1061ms
Ping to www.google.com... Get pong in 8.9211ms
Ping to www.google.com... Get pong in 7.9899ms
```

3.3.3 原理分析

Pinger类定义了很多有用的方法，初始化时创建了几个变量，其值由用户指定，或者有默认值，如下所示：

- target_host：要查验的目标主机；
- count：查验次数；
- timeout：这个值决定何时终止未完成的查验操作。

在send_ping()方法中获取了目标主机的DNS主机名，然后使用struct模块创建了一个ICMP_ECHO_REQUEST数据包。在这个方法中一定要使用do_checksum()方法检查数据的完整性。do_checksum()方法接收一个源字符串，经过处理之后生成一个特有的校验和。在接收端，receive_pong()方法在未到达超时时间之前一直等待响应，或者直接接收响应，然后抓取ICMP响应首部，对比数据包ID，再计算请求-响应循环的延迟时间。

3.4 等待远程网络服务上线

有时，在网络服务恢复的过程中，可以运行一个脚本检查服务器何时再次上线。

3.4.1 实战演练

我们可以编写一个客户端，一直等待某个网络服务上线，或者只等待一段时间。在这个示例中，默认情况下我们检查的是本地主机中的一个Web服务器。如果你指定了其他远程主机或端口，这个脚本会使用你提供的信息。

代码清单3-3展示了如何等待远程网络服务上线，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 3
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import socket
```

```
import errno
from time import time as now

DEFAULT_TIMEOUT = 120
DEFAULT_SERVER_HOST = 'localhost'
DEFAULT_SERVER_PORT = 80

class NetServiceChecker(object):
    """ Wait for a network service to come online"""
    def __init__(self, host, port, timeout=DEFAULT_TIMEOUT):
        self.host = host
        self.port = port
        self.timeout = timeout
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    def end_wait(self):
        self.sock.close()

    def check(self):
        """ Check the service """
        if self.timeout:
            end_time = now() + self.timeout

        while True:
            try:
                if self.timeout:
                    next_timeout = end_time - now()
                    if next_timeout < 0:
                        return False
                    else:
                        print "setting socket next timeout %ss" %round(next_timeout)
                        self.sock.settimeout(next_timeout)
                self.sock.connect((self.host, self.port))
            # handle exceptions
            except socket.timeout, err:
                if self.timeout:
                    return False
            except socket.error, err:
                print "Exception: %s" %err
            else: # if all goes well
                self.end_wait()
                return True

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Wait for Network Service')
    parser.add_argument('--host', action="store", dest="host",
default=DEFAULT_SERVER_HOST)
    parser.add_argument('--port', action="store", dest="port", type=int,
default=DEFAULT_SERVER_PORT)
    parser.add_argument('--timeout', action="store", dest="timeout", type=int,
default=DEFAULT_TIMEOUT)
    given_args = parser.parse_args()
```

```
    host, port, timeout = given_args.host, given_args.port, given_args.timeout
    service_checker = NetServiceChecker(host, port, timeout=timeout)
    print "Checking for network service %s:%s ..." %(host, port)
    if service_checker.check():
        print "Service is available again!"
```

如果在你的设备上运行着一个Web服务器，例如Apache，运行这个脚本后会看到如下输出：

```
$ python 3_3_wait_for_remote_service.py
Waiting for network service localhost:80 ...
setting socket next timeout 120.0s
Service is available again!
```

现在停止Apache进程，再运行这个脚本，然后重启Apache。此时看到的输出会有所不同，在我的设备上，输出如下：

```
Exception: [Errno 103] Software caused connection abort
setting socket next timeout 104.189137936
Exception: [Errno 111] Connection refused
setting socket next timeout 104.186291933
Exception: [Errno 103] Software caused connection abort
setting socket next timeout 104.186164856
Service is available again!
```

下面的截图展示了等待Apache Web服务器上线的过程：



3.4.2　原理分析

上述脚本使用argparse模块接收用户的输入，处理主机名、端口和超时时间。超时时间指等待所需网络服务的时间。这个脚本创建了一个NetServiceChecker类实例，然后调用check()方法。这个方法计算等待的最后结束时间，并使用套接字的settimeout()方法控制每次循环的结束时间，即next_timeout。然后check()方法调用套接字的connect()方法在超时时间到达之前测试所需的网络服务是否可用。check()方法还能捕获套接字超时异常，以及比较套接字超时时间和用户指定的超时时间。

3.5 枚举设备中的接口

在Python中列出设备中的网络接口并不难。有很多第三方库可以使用，只需几行代码即可。不过，我们来看一下如何只使用套接字调用完成这一操作。

3.5.1 准备工作

这个攻略需要在Linux设备中运行。若想列出可用的网络接口，可以执行下面的命令：

```
$ /sbin/ifconfig
```


3.5.2 实战演练

代码清单3-4展示了如何列出网络接口，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 3
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import sys
import socket
import fcntl
import struct
import array

SIOCGIFCONF = 0x8912 #from C library sockios.h
STUCT_SIZE_32 = 32
STUCT_SIZE_64 = 40
PLATFORM_32_MAX_NUMBER =  2**32
DEFAULT_INTERFACES = 8

def list_interfaces():
    interfaces = []
    max_interfaces = DEFAULT_INTERFACES
    is_64bits = sys.maxsize > PLATFORM_32_MAX_NUMBER
    struct_size = STUCT_SIZE_64 if is_64bits else STUCT_SIZE_32
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    while True:
        bytes = max_interfaces * struct_size
        interface_names = array.array('B', '\0' * bytes)
        sock_info = fcntl.ioctl(
            sock.fileno(),
            SIOCGIFCONF,
            struct.pack('iL', bytes, interface_names.buffer_info()[0])
        )
        outbytes = struct.unpack('iL', sock_info)[0]
        if outbytes == bytes:
```

```
            max_interfaces *= 2
        else:
            break
    namestr = interface_names.tostring()
    for i in range(0, outbytes, struct_size):
        interfaces.append((namestr[i:i+16].split('\0', 1)[0]))
    return interfaces

if __name__ == '__main__':
    interfaces = list_interfaces()
    print "This machine has %s network interfaces: %s." %(len(interfaces), interfaces)
```

上述脚本能列出网络接口，输出结果如下：

```
$ python 3_4_list_network_interfaces.py
This machine has 2 network interfaces: ['lo', 'eth0'].
```

3.5.3　原理分析

这个攻略使用低层套接字特性找出系统中的接口。list_interfaces()方法创建一个套接字对象，通过处理这个对象找到网络接口信息，做法是调用fnctl模块中的ioctl()方法。fnctl模块用到了一些Unix程序，例如fnctl()。这个接口在底层的文件描述符套接字上执行I/O控制操作。文件描述符通过在套接字对象上调用fileno()方法获取。

ioctl()方法的其他参数包括：C套接字库中定义的常量SIOCGIFADDR，以及使用struct模块中的pack()函数生成的数据结构。数据结构中指定的内存地址保存在变量interface_names中，经修改后作为ioctl()方法的结果返回。从ioctl()方法的返回结果sock_info中取出数据后，如果大小和变量bytes相等，则将网络接口的数量翻倍。为了防止之前假设的接口数量不正确，这个操作要在一个while循环中完成，以便找出所有接口。

接口的名字从变量interface_names的字符串形式中提取，先读取这个变量的指定字段，然后再把获得的值添加到接口列表的末尾。在list_interfaces()函数的最后，返回这个接口列表。

3.6　找出设备中某个接口的 IP 地址

在Python网络应用程序中可能需要找出某个网络接口的IP地址。

3.6.1　准备工作

这个攻略是Linux专用的。有一些Python模块经过特别设计，为Windows和Mac平台提供了类似的功能。例如，http://sourceforge.net/projects/pywin32/是专为Windows实现的库。

3.6.2　实战演练

你可以使用fnctl模块在你的设备中查询IP地址。

代码清单3-5展示了如何在你的设备中找出指定接口的IP地址，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 3
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import sys
import socket
import fcntl
import struct
import array

def get_ip_address(ifname):
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    return socket.inet_ntoa(fcntl.ioctl(
        s.fileno(),
        0x8915,  # SIOCGIFADDR
        struct.pack('256s', ifname[:15])
    )[20:24])

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Python networking utils')
    parser.add_argument('--ifname', action="store", dest="ifname", required=True)
    given_args = parser.parse_args()
    ifname = given_args.ifname
    print "Interface [%s] --> IP: %s" %(ifname, get_ip_address(ifname))
```

这个脚本的输出只有一行，如下所示：

```
$ python 3_5_get_interface_ip_address.py --ifname=eth0
Interface [eth0] --> IP: 10.0.2.15
```

3.6.3　原理分析

这个攻略和前一个攻略类似。上述脚本接收一个命令行参数：要查询的IP地址的网络接口名。get_ip_address()函数创建一个套接字对象，然后调用fnctl.ioctl()函数利用这个套接字对象查询IP信息。注意，socket.inet_ntoa()函数的作用是，把二进制数据转换成我们熟悉的人类可读的点分格式。

3.7　探测设备中的接口是否开启

如果设备中有多个网络接口，在使用某个接口前你需要知道它的状态，例如，这个接口是否

开启。这样才能保证把命令传递给处于激活状态的接口。

3.7.1　准备工作

这个攻略是为Linux设备编写的，因此无法在Windows或Mac主机上运行。这个攻略用到了一个著名的网络扫描工具——nmap。在nmap的网站http://nmap.org/中可以了解更多信息。

运行这个攻略还需要python-nmap模块，可使用pip安装，如下所示：

```
$ pip install python-nmap
```

3.7.2　实战演练

我们可以创建一个套接字对象，然后获取接口的IP地址，再使用任何一种扫描技术探测接口的状态。

代码清单3-6展示了如何探测网络接口的状态，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 3
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import socket
import struct
import fcntl
import nmap

SAMPLE_PORTS = '21-23'

def get_interface_status(ifname):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    ip_address = socket.inet_ntoa(fcntl.ioctl(
        sock.fileno(),
        0x8915, #SIOCGIFADDR, C socket library sockios.h
        struct.pack('256s', ifname[:15])
    )[20:24])
    nm = nmap.PortScanner()
    nm.scan(ip_address, SAMPLE_PORTS)
    return nm[ip_address].state()

if  __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Python networking utils')
    parser.add_argument('--ifname', action="store", dest="ifname", required=True)
    given_args = parser.parse_args()
    ifname = given_args.ifname
    print "Interface [%s] is: %s" %(ifname, get_interface_status(ifname))
```

如果运行这个脚本查询eth0的状态，会看到类似下面的输出：

```
$ python 3_6_find_network_interface_status.py --ifname=eth0
Interface [eth0] is: up
```

3.7.3　原理分析

这个攻略从命令行中读取接口名，然后将其传给get_interface_status()函数。这个函数通过处理一个UDP套接字对象找到该接口的IP地址。

这个攻略需要第三方模块nmap的支持。我们可以使用pip从PyPI上安装这个模块。nmap扫描的实例nm，是通过调用PortScanner()创建的。初步扫描本地IP后就能获取对应网络接口的状态。

3.8　检测网络中未开启的设备

如果有人给你网络中一些设备的IP地址，让你编写一个脚本定期找出哪些主机未开启，你可以编写一个网络扫描类型的程序，而无需在目标主机电脑中安装任何软件。

3.8.1　准备工作

这个攻略需要安装Scapy库（2.2以上版本），下载地址为http://www.secdev.org/projects/scapy/files/scapy-latest.zip。

3.8.2　实战演练

我们可以使用成熟的第三方网络分析库Scapy启动ICMP扫描。因为我们要定期运行这个脚本，所以需要用到Python中的sched模块，安排扫描任务。

代码清单3-7展示了如何检测未开启的设备，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 3
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.
# This recipe requires scapy-2.2.0 or higher

import argparse
import time
import sched
from scapy.all import sr, srp, IP, UDP, ICMP, TCP, ARP, Ether
RUN_FREQUENCY = 10
scheduler = sched.scheduler(time.time, time.sleep)
```

```
def detect_inactive_hosts(scan_hosts):
    """
    Scans the network to find scan_hosts are live or dead
    scan_hosts can be like 10.0.2.2-4 to cover range.
    See Scapy docs for specifying targets.
    """
    global scheduler
    scheduler.enter(RUN_FREQUENCY, 1, detect_inactive_hosts, (scan_hosts, ))
    inactive_hosts = []
    try:
        ans, unans = sr(IP(dst=scan_hosts)/ICMP(), retry=0, timeout=1)
        ans.summary(lambda(s,r) : r.sprintf("%IP.src% is alive"))
        for inactive in unans:
            print "%s is inactive" %inactive.dst
            inactive_hosts.append(inactive.dst)

        print "Total %d hosts are inactive" %(len(inactive_hosts))
    except KeyboardInterrupt:
        exit(0)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Python networking utils')
    parser.add_argument('--scan-hosts', action="store", dest="scan_hosts",
required=True)
    given_args = parser.parse_args()
    scan_hosts = given_args.scan_hosts
    scheduler.enter(1, 1, detect_inactive_hosts, (scan_hosts, ))
    scheduler.run()
```

这个脚本的输出如下面的命令行所示：

```
$ sudo python 3_7_detect_inactive_machines.py --scan-hosts=10.0.2.2-4
Begin emission:
.*...Finished to send 3 packets.
.
Received 6 packets, got 1 answers, remaining 2 packets
10.0.2.2 is alive
10.0.2.4 is inactive
10.0.2.3 is inactive
Total 2 hosts are inactive
Begin emission:
*.Finished to send 3 packets.
Received 3 packets, got 1 answers, remaining 2 packets
10.0.2.2 is alive
10.0.2.4 is inactive
10.0.2.3 is inactive
Total 2 hosts are inactive
```

3.8.3　原理分析

上述脚本先从命令行中读取一组网络主机的地址，保存到变量scan_hosts中，然后创建一

个日程表，每隔一秒运行一次detect_inactive_hosts()函数。detect_inactive_hosts()函数的参数是scan_hosts，该函数调用了Scapy库的sr()函数。

detect_inactive_hosts()函数再次调用schedule.enter()函数，以安排自己10秒钟之后再次运行。如此一来，我们就能定期执行扫描任务了。

Scapy库的sr()函数接收的参数分别是IP、协议和一些扫描控制信息。在这个脚本中，把scan_hosts传给IP()方法，作为扫描的目标主机，协议指定为ICMP。协议还可使用TCP或UDP。我们没有指定重试一次并把超时时间设为一秒，以便提升脚本的运行速度。你可以自己尝试，找到符合需求的选项值。

扫描函数sr()在一个元组中返回有应答的主机和无应答的主机。我们获取了无应答的主机，构建成一个列表，然后打印出来。

3.9　使用相连的套接字执行基本的进程间通信

有时，两个脚本要通过两个进程彼此通信。在Unix/Linux中，有一个概念叫作“相连的套接字”，即socketpair。这一节对此做些实验。

3.9.1　准备工作

这个脚本为Unix/Linux主机而编写，不适合在Windows/Mac中运行。

3.9.2　实战演练

我们要在test_socketpair()函数中编写几行代码，测试套接字的socketpair()函数。

代码清单3-8是一个socketpair用法示例，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 3
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import socket
import os

BUFSIZE = 1024

def test_socketpair():
    """ Test Unix socketpair"""
    parent, child = socket.socketpair()
```

```
    pid = os.fork()
    try:
        if pid:
            print "@Parent, sending message..."
            child.close()
            parent.sendall("Hello from parent!")
            response = parent.recv(BUFSIZE)
            print "Response from child:", response
            parent.close()

        else:
            print "@Child, waiting for message from parent"
            parent.close()
            message = child.recv(BUFSIZE)
            print "Message from parent:", message
            child.sendall("Hello from child!!")
            child.close()
    except Exception, err:
        print "Error: %s" %err

if __name__ == '__main__':
    test_socketpair()
```

上述脚本的输出如下所示：

```
$ python 3_8_ipc_using_socketpairs.py
@Parent, sending message...
@Child, waiting for message from parent
Message from parent: Hello from parent!
Response from child: Hello from child!!
```

3.9.3　原理分析

socket.socketpair()函数返回的是两个相连的套接字对象，这里我们把其中一个称为父套接字，另一个称为子套接字。我们调用os.fork()方法派生出了另一个进程，其返回结果是父进程的ID。在各个进程中，先把另一个进程中的套接字关闭，然后在当前进程中的套接字上调用sendall()方法交换消息。在try-except块中如果出现异常，就把错误打印出来。

3.10　使用 Unix 域套接字执行进程间通信

有时使用Unix域套接字（Unix Domain Socket，简称UDS）处理两个进程之间的通信更方便。在Unix中，一切都是文件。如果你需要一个这种进程间通信的例子，这个攻略可以给你一些帮助。

3.10.1　实战演练

我们要启动一个UDS服务器，绑定到一个文件系统路径上。然后启动一个UDS客户端，使用

相同的路径和服务器通信。

代码清单3-9a是一个Unix域套接字服务器，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 3
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import socket
import os
import time

SERVER_PATH = "/tmp/python_unix_socket_server"

def run_unix_domain_socket_server():
    if os.path.exists(SERVER_PATH):
        os.remove( SERVER_PATH )

    print "starting unix domain socket server."
    server = socket.socket( socket.AF_UNIX, socket.SOCK_DGRAM )
    server.bind(SERVER_PATH)

    print "Listening on path: %s" %SERVER_PATH
    while True:
        datagram = server.recv( 1024 )
        if not datagram:
            break
        else:
            print "-" * 20
            print datagram
        if "DONE" == datagram:
            break
    print "-" * 20
    print "Server is shutting down now..."
    server.close()
    os.remove(SERVER_PATH)
    print "Server shutdown and path removed."

if __name__ == '__main__':
    run_unix_domain_socket_server()
```

代码清单3-9b是一个UDS客户端，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 3
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import socket
import sys
```

```
SERVER_PATH = "/tmp/python_unix_socket_server"

def run_unix_domain_socket_client():
    """ Run "a Unix domain socket client """
    sock = socket.socket(socket.AF_UNIX, socket.SOCK_DGRAM)

    # Connect the socket to the path where the server is listening
    server_address = SERVER_PATH
    print "connecting to %s" % server_address
    try:
        sock.connect(server_address)
    except socket.error, msg:
        print >>sys.stderr, msg
        sys.exit(1)

    try:
        message = "This is the message.  This will be echoed back!"
        print  "Sending [%s]" %message
        sock.sendall(message)
        amount_received = 0
        amount_expected = len(message)

        while amount_received < amount_expected:
            data = sock.recv(16)
            amount_received += len(data)
            print >>sys.stderr, "Received [%s]" % data

    finally:
        print "Closing client"
        sock.close()

if __name__ == '__main__':
    run_unix_domain_socket_client()
```

服务器的输出如下所示：

```
$ python 3_9a_unix_domain_socket_server.py
starting unix domain socket server.
Listening on path: /tmp/python_unix_socket_server
--------------------
This is the message.  This will be echoed back!
```

客户端的输出如下所示：

```
$ python 3_9b_unix_domain_socket_client.py
connecting to /tmp/python_unix_socket_server
Sending [This is the message.  This will be echoed back!]
```

3.10.2　原理分析

我们为UDS客户端和服务器定义了一个共用的路径，二者都用这个路径连接和监听。

在服务器的代码中，如果前一次运行脚本后路径仍然存在，就将其删除。然后创建一个Unix数据报套接字，绑定到指定的路径上，监听进入的连接。在数据处理循环中，使用recv()方法获取客户端发出的数据并打印到屏幕上。

客户端代码直接打开一个Unix数据报套接字,连接共用的服务器地址。客户端调用sendall()方法向服务器发送一个消息，然后等待这些消息返回，再打印出来。

3.11 确认你使用的 Python 是否支持 IPv6 套接字

IP第6版（IPv6）在业内越来越多地被用来开发新型应用。如果你想编写一个IPv6应用程序，首先要知道你的设备是否支持IPv6。在Linux/Unix中，可通过下面的命令确认：

```
$ cat /proc/net/if_inet6
00000000000000000000000000000001 01 80 10 80       lo
fe800000000000000a0027fffe950d1a 02 40 20 80     eth0
```

使用Python脚本也可以检查你的设备是否支持IPv6，以及所安装的Python是否支持。

3.11.1 准备工作

在这个攻略中，要使用pip安装一个Python第三方库，netifaces，如下所示：

```
$ pip install netifaces
```

3.11.2 实战演练

我们可以使用第三方库netifaces确认你的设备是否支持IPv6。我们要调用这个库中的interfaces()函数，列出系统中的所有接口。

代码清单3-10是检查设备是否支持IPv6的Python脚本，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 3
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.
#IPv6  test in Unix  commandline: $cat /proc/net/if_inet6

import socket
import argparse
import netifaces as ni

def inspect_ipv6_support():
    """ Find the ipv6 address"""
    print "IPV6 support built into Python: %s" %socket.has_ipv6
```

```
    ipv6_addr = {}
    for interface in ni.interfaces():
        all_addresses = ni.ifaddresses(interface)
        print "Interface %s:" %interface
        for family,addrs in all_addresses.iteritems():
            fam_name = ni.address_families[family]
            print '  Address family: %s' % fam_name
            for addr in addrs:
                if fam_name == 'AF_INET6':
                    ipv6_addr[interface] = addr['addr']
                print     '    Address  : %s' % addr['addr']
                nmask = addr.get('netmask', None)
                if nmask:
                    print '    Netmask  : %s' % nmask
                bcast = addr.get('broadcast', None)
                if bcast:
                    print '    Broadcast: %s' % bcast
    if ipv6_addr:
        print "Found IPv6 address: %s" %ipv6_addr
    else:
        print "No IPv6 interface found!"

if __name__ == '__main__':
    inspect_ipv6_support()
```

这个脚本的输出如下所示：

```
$ python 3_10_check_ipv6_support.py
IPV6 support built into Python: True
Interface lo:
  Address family: AF_PACKET
    Address  : 00:00:00:00:00:00
  Address family: AF_INET
    Address  : 127.0.0.1
    Netmask  : 255.0.0.0
  Address family: AF_INET6
    Address : ::1
    Netmask  : ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff
Interface eth0:
  Address family: AF_PACKET
    Address  : 08:00:27:95:0d:1a
    Broadcast: ff:ff:ff:ff:ff:ff
  Address family: AF_INET
    Address  : 10.0.2.15
    Netmask  : 255.255.255.0
    Broadcast: 10.0.2.255
  Address family: AF_INET6
    Address  : fe80::a00:27ff:fe95:d1a
    Netmask  : ffff:ffff:ffff:ffff::
Found IPv6 address: {'lo': '::1', 'eth0': 'fe80::a00:27ff:fe95:d1a'}
```

3.11.3　原理分析

检查设备是否支持IPv6的函数inspect_ipv6_support()首先使用socket.has_ipv6检查编译Python时是否加入了IPv6支持。然后调用netifaces模块中的interfaces()函数，列出所有接口。调用ifaddresses()方法时如果传入了一个网络接口，会返回这个接口的所有IP地址。然后从中提取不同的IP相关信息，例如协议族、地址、网络掩码和广播地址。如果协议族匹配AF_INET6，就把网络接口的地址添加到IPv6_address字典中。

3.12　从 IPv6 地址中提取 IPv6 前缀

在IPv6应用中，你要从IPv6地址中找出前缀信息。注意，按照RFC 3513的定义，前面的64位IPv6地址由全网路由前缀和子网ID组成。通常使用一个较短的前缀（例如/48），可以定义很多更长、更具体的前缀（例如/64）。使用Python脚本可以更方便的生成前缀信息。

3.12.1　实战演练

我们可以使用第三方库netifaces和netaddr找出IPv6地址中的IPv6前缀，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 3
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import socket
import netifaces as ni
import netaddr as na

def extract_ipv6_info():
    """ Extracts IPv6 information"""
    print "IPV6 support built into Python: %s" %socket.has_ipv6
    for interface in ni.interfaces():
        all_addresses = ni.ifaddresses(interface)
        print "Interface %s:" %interface
        for family,addrs in all_addresses.iteritems():
            fam_name = ni.address_families[family]
            #print '  Address family: %s' % fam_name
            for addr in addrs:
                if fam_name == 'AF_INET6':
                    addr = addr['addr']
                    has_eth_string = addr.split("%eth")
                    if has_eth_string:
                        addr = addr.split("%eth")[0]
                    print "    IP Address: %s" %na.IPNetwork(addr)
                    print "    IP Version: %s" %na.IPNetwork(addr).version
                    print "    IP Prefix length: %s" %na.IPNetwork(addr).prefixlen
                    print "    Network: %s" %na.IPNetwork(addr).network
```

```
                print "    Broadcast: %s" %na.IPNetwork(addr).broadcast

if __name__ == '__main__':
    extract_ipv6_info()
```

这个脚本的输出如下所示：

```
$ python 3_11_extract_ipv6_prefix.py
IPV6 support built into Python: True
Interface lo:
    IP Address: ::1/128
    IP Version: 6
    IP Prefix length: 128
    Network: ::1
    Broadcast: ::1
Interface eth0:
    IP Address: fe80::a00:27ff:fe95:d1a/128
    IP Version: 6
    IP Prefix length: 128
    Network: fe80::a00:27ff:fe95:d1a
    Broadcast: fe80::a00:27ff:fe95:d1a
```

3.12.2　原理分析

Python的netifaces库使用interfaces()和ifaddresses()两个函数获取网络接口的IPv6地址。处理网络地址时使用netaddr模块特别方便。这个模块中的IPNetwork()类构造方法会提供一个IPv4或IPv6地址，并计算出前缀、网络地址和广播地址。这些信息从IPNetwork()类实例的version、prefixlen、network和broadcast属性中获取。

3.13　编写一个 IPv6 回显客户端/服务器

你要编写一个支持IPv6的服务器或客户端，才能知道它和IPv4版有何区别。

3.13.1　实战演练

这里使用的方案和编写IPv4回显客户端/服务器一样。唯一重要的区别是，使用IPv6信息创建套接字的方法。

代码清单3-11a是IPv6回显服务器，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 3
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
```

```
import socket
import sys

HOST = 'localhost'

def echo_server(port, host=HOST):
    """Echo server using IPv6 """
    for result in socket.getaddrinfo(host, port, socket.AF_UNSPEC, socket.SOCK_STREAM,
0, socket.AI_PASSIVE):
        af, socktype, proto, canonname, sa = result
        try:
            sock = socket.socket(af, socktype, proto)
        except socket.error, err:
            print "Error: %s" %err

        try:
            sock.bind(sa)
            sock.listen(1)
            print "Server lisenting on %s:%s" %(host, port)
        except socket.error, msg:
            sock.close()
            continue
        break
        sys.exit(1)
    conn, addr = sock.accept()
    print 'Connected to', addr
    while True:
        data = conn.recv(1024)
        print "Received data from the client: [%s]" %data
        if not data: break
        conn.send(data)
        print "Sent data echoed back to the client: [%s]" %data
    conn.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='IPv6 Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int,
required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_server(port)
```

代码清单3-11b是IPv6回显客户端，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 3
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import socket
import sys
```

```
HOST = 'localhost'
BUFSIZE = 1024

def ipv6_echo_client(port, host=HOST):
    for res in socket.getaddrinfo(host, port, socket.AF_UNSPEC, socket.SOCK_STREAM):
        af, socktype, proto, canonname, sa = res
        try:
            sock = socket.socket(af, socktype, proto)
        except socket.error, err:
            print "Error:%s" %err
        try:
            sock.connect(sa)
        except socket.error, msg:
            sock.close()
            continue
    if sock is None:
        print 'Failed to open socket!'
        sys.exit(1)
    msg = "Hello from ipv6 client"
    print "Send data to server: %s" %msg
    sock.send(msg)
    while True:
        data = sock.recv(BUFSIZE)
        print 'Received from server', repr(data)
        if not data:
            break
    sock.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='IPv6 socket client example')
    parser.add_argument('--port', action="store", dest="port", type=int,
required=True)
    given_args = parser.parse_args()
    port = given_args.port
    ipv6_echo_client(port)
```

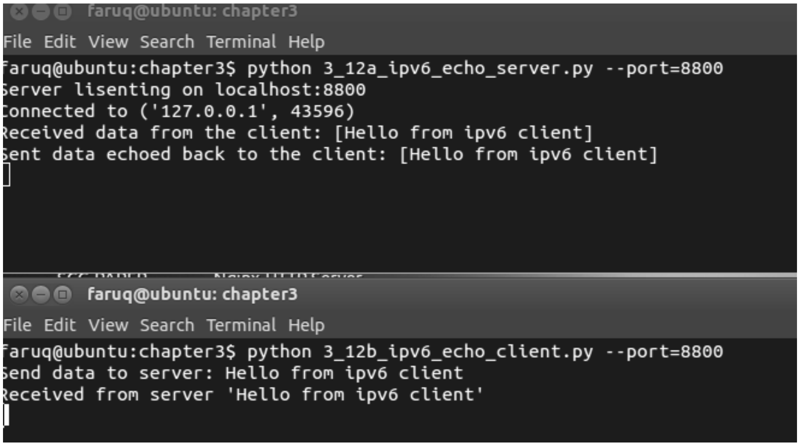
服务器的输出如下：

```
$ python 3_12a_ipv6_echo_server.py --port=8800
Server lisenting on localhost:8800
Connected to ('127.0.0.1', 35034)
Received data from the client: [Hello from ipv6 client]
Sent data echoed back to the client: [Hello from ipv6 client]
```

客户端的输出如下：

```
$ python 3_12b_ipv6_echo_client.py --port=8800
Send data to server: Hello from ipv6 client
Received from server 'Hello from ipv6 client'
```

下面的截图展示了IPv6客户端和服务器之间的交互：



3.13.2 原理分析

IPv6回显服务器首先调用`socket.getaddrinfo()`获取自身的IPv6信息。注意，创建TCP套接字时指定的协议是`AF_UNSPEC`。得到的信息是有五个值的元组。创建服务器套接字时用到了其中三个信息：地址族、套接字类型和协议。然后把套接字绑定到元组中保存的套接字地址上，监听并接受进入的连接。建立连接后，服务器接收客户端发来的数据，然后回显给客户端。

在客户端代码中，我们创建了一个兼容IPv6的客户端套接字实例，然后在这个实例上调用`send()`方法发送数据，再调用`recv()`方法获取服务器回显的数据。

第 4 章 HTTP协议网络编程

本章攻略：

- 从HTTP服务器下载数据
- 在你的设备中伺服HTTP请求
- 访问网站后提取cookie信息
- 提交网页表单
- 通过代理服务器发送Web请求
- 使用HEAD请求检查网页是否存在
- 把客户端伪装成Mozilla Firefox
- 使用HTTP压缩节省Web请求消耗的带宽
- 编写一个支持断点续传功能的HTTP容错客户端
- 使用Python和OpenSSL编写一个简单的HTTPS服务器

4.1 简介

本章介绍Python HTTP网络库和一些第三方库的功能。例如，以一种更友好、更简洁的方式处理HTTP请求的`requests`库。其中有一个攻略用到了`OpenSSL`库，创建支持SSL的Web服务器。

多个攻略都介绍了HTTP协议的很多常规特性，例如使用`POST`请求提交网页表单、处理首部信息和使用压缩等。

4.2 从 HTTP 服务器下载数据

你可能想要编写一个简单的HTTP客户端，通过原生的HTTP协议从任意的Web服务器上下载一些数据。这是自己开发HTTP浏览器的第一步。

4.2.1 实战演练

我们要使用Python编写的微型浏览器访问www.python.org。这个浏览器使用Python中的httplib模块编写。

代码清单4-1说明了如何编写一个简单的HTTP客户端，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 4
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import httplib

REMOTE_SERVER_HOST = 'www.python.org'
REMOTE_SERVER_PATH = '/'

class HTTPClient:

    def __init__(self, host):
        self.host = host

    def fetch(self, path):
        http = httplib.HTTP(self.host)

        # Prepare header
        http.putrequest("GET", path)
        http.putheader("User-Agent", __file__)
        http.putheader("Host", self.host)
        http.putheader("Accept", "*/*")
        http.endheaders()

        try:
            errcode, errmsg, headers = http.getreply()
        except Exception, e:
            print "Client failed error code: %s message:%s headers:%s" %(errcode, 
errmsg, headers)
        else:
            print "Got homepage from %s" %self.host

        file = http.getfile()
        return file.read()

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='HTTP Client Example')
    parser.add_argument('--host', action="store", dest="host", 
default=REMOTE_SERVER_HOST)
    parser.add_argument('--path', action="store", dest="path", 
default=REMOTE_SERVER_PATH)
    given_args = parser.parse_args()
    host, path = given_args.host, given_args.path
```

4

```
    client = HTTPClient(host)
    print client.fetch(path)
```

这个攻略默认从www.python.org中获取一个网页。运行这个脚本时可以指定主机和路径参数，也可以不指定。运行脚本后会看到如下输出：

```
$  python 4_1_download_data.py --host=www.python.org
Got homepage from www.python.org
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.og/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <title>Python Programming Language &ndash; Official Website</title>
....
```

如果运行脚本时指定的路径不存在，会显示如下的服务器响应：

```
$ python 4_1_download_data.py --host='www.python.org' --path='/not-exist'
Got homepage from www.python.org
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <title>Page Not Found</title>
  <meta name="keywords" content="Page Not Found" />
  <meta name="description" content="Page Not Found" />
```

4.2.2　原理分析

这个攻略使用Python的内置库`httplib`，定义了一个`HTTPClient`类，从远程主机上获取数据。在`fetch()`方法中使用`HTTP()`函数及其他辅助函数（例如`putrequest()`和`putheader()`）创建了一个虚拟的HTTP客户端，首先指定一个`GET/path`字符串，然后设定用户代理，其值为当前脚本（`__file__`）。

发起请求的`getreply()`方法放在一个`try-except`块中。响应通过`getfile()`方法获取，然后读取数据流中的内容。

4.3　在你的设备中伺服 HTTP 请求

你可能想编写一个自己的Web服务器，处理客户端请求，返回一个简单的欢迎消息。

4.3.1　实战演练

Python集成了一个非常简单的Web服务器，可以在命令行中启动，如下所示：

```
$ python -m SimpleHTTPServer 8080
```

执行这个命令后会在端口8080上启动一个HTTP Web服务器。通过在浏览器中输入http://localhost:8080，可以访问这个服务器。你将看到的是运行上述命令时所在文件夹里的内容。如果这个文件夹中有能被Web服务器识别的索引文件，例如index.html，在浏览器中就会显示这个文件的内容。如果你想完全掌控Web服务器，就得启动自己定制的HTTP服务器。

代码清单4-2是这个定制的HTTP Web服务器，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 4
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import sys
from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer

DEFAULT_HOST = '127.0.0.1'
DEFAULT_PORT = 8800

class RequestHandler(BaseHTTPRequestHandler):
    """ Custom request handler"""

    def do_GET(self):
        """ Handler for the GET requests """
        self.send_response(200)
        self.send_header('Content-type','text/html')
        self.end_headers()
        # Send the message to browser
        self.wfile.write("Hello from server!")
        return

class CustomHTTPServer(HTTPServer):
    "A custom HTTP server"
    def __init__(self, host, port):
        server_address = (host, port)
        HTTPServer.__init__(self, server_address, RequestHandler)

def run_server(port):
    try:
        server= CustomHTTPServer(DEFAULT_HOST, port)
        print "Custom HTTP server started on port: %s" % port
        server.serve_forever()
    except Exception, err:
        print "Error:%s" %err
    except KeyboardInterrupt:
        print "Server interrupted and is shutting down..."
        server.socket.close()

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Simple HTTP Server Example')
```

```
    parser.add_argument('--port', action="store", dest="port", type=int,
default=DEFAULT_PORT)
    given_args = parser.parse_args()
    port = given_args.port
    run_server(port)
```

下面的截图是一个简单的HTTP服务器：



运行这个Web服务器，然后在浏览器中访问，会看到浏览器中显示了一行文本“Hello from server!”，如下所示：

```
$ python 4_2_simple_http_server.py --port=8800
Custom HTTP server started on port: 8800
localhost - - [18/Apr/2013 13:39:33] "GET / HTTP/1.1" 200 -
localhost - - [18/Apr/2013 13:39:33] "GET /favicon.ico HTTP/1.1" 200
```

4.3.2　原理分析

在这个攻略中，我们定义了`CustomHTTPServer`类，它继承自`HTTPServer`类。在`CustomHTTPServer`类的构造方法中，设定了服务器地址和用户输入的端口号，还用到了`RequestHandler`类。客户端连到服务器上时，服务器就通过`RequestHandler`类处理请求。

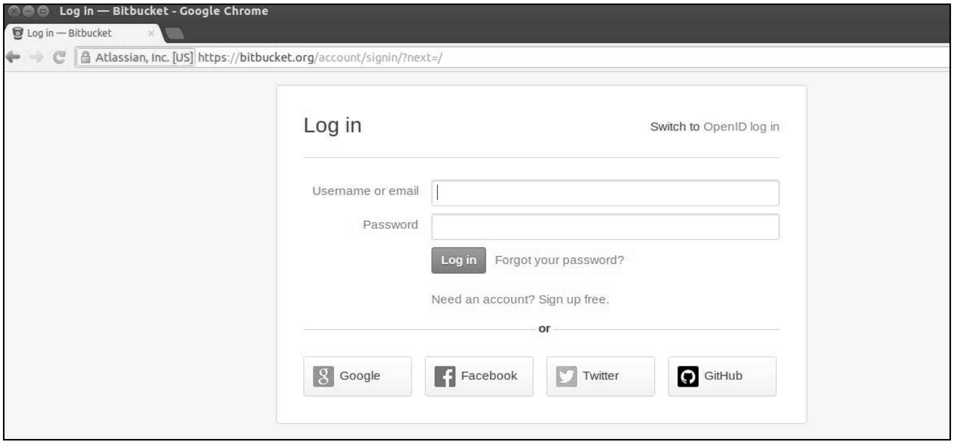
`RequestHandler`类定义了处理客户端`GET`请求的方法。这个方法向客户端发送一个HTTP首部（状态码200），然后使用`write()`方法返回一个成功消息“Hello from server!”。

4.4　访问网站后提取 cookie 信息

很多网站使用cookie在你的本地硬盘中存储各种信息。你可能想要查看cookie中保存的信息，或者使用cookie自动登录网站。

4.4.1　实战演练

假设我们要登录流行的代码分享网站www.bitbucket.org，我们要在登录页面（https://bitbucket.org/ account/signin/?next=/）提交登录信息。登录页面的截图如下所示：



我们要记下表单中几个字段的ID，然后决定提交哪些虚拟值。我们首先要访问登录页面，再访问首页，查看在cookie中保存了什么。

代码清单4-3说明了如何提取cookie信息，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 4
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import cookielib
import urllib
import urllib2

ID_USERNAME = 'id_username'
ID_PASSWORD = 'id_password'
USERNAME = 'you@email.com'
PASSWORD = 'mypassword'
LOGIN_URL = 'https://bitbucket.org/account/signin/?next=/'
NORMAL_URL = 'https://bitbucket.org/'

def extract_cookie_info():
    """ Fake login to a site with cookie"""
    # setup cookie jar
    cj = cookielib.CookieJar()
    login_data = urllib.urlencode({ID_USERNAME : USERNAME,
                                   ID_PASSWORD : PASSWORD})
    # create url opener
    opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cj))
```

```
    resp = opener.open(LOGIN_URL, login_data)

    # send login info
    for cookie in cj:
        print "----First time cookie: %s --> %s" %(cookie.name, cookie.value)

    print "Headers: %s" %resp.headers

    # now access without any login info
    resp = opener.open(NORMAL_URL)
    for cookie in cj:
        print "++++Second time cookie: %s --> %s" %(cookie.name, cookie.value)

    print "Headers: %s" %resp.headers

if __name__ == '__main__':
    extract_cookie_info()
```

运行这个脚本后得到的输出如下：

```
$ python 4_3_extract_cookie_information.py
----First time cookie: bb_session --> aed58dde1228571bf60466581790566d
Headers: Server: nginx/1.2.4
Date: Sun, 05 May 2013 15:13:56 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 21167
Connection: close
X-Served-By: bitbucket04
Content-Language: en
X-Static-Version: c67fb01467cf
Expires: Sun, 05 May 2013 15:13:56 GMT
Vary: Accept-Language, Cookie
Last-Modified: Sun, 05 May 2013 15:13:56 GMT
X-Version: 14f9c66ad9db
ETag: "3ba81d9eb350c295a453b5ab6e88935e"
X-Request-Count: 310
Cache-Control: max-age=0
Set-Cookie: bb_session=aed58dde1228571bf60466581790566d; expires=Sun, 19-
May-2013 15:13:56 GMT; httponly; Max-Age=1209600; Path=/; secure

Strict-Transport-Security: max-age=2592000
X-Content-Type-Options: nosniff

++++Second time cookie: bb_session --> aed58dde1228571bf60466581790566d
Headers: Server: nginx/1.2.4
Date: Sun, 05 May 2013 15:13:57 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 36787
Connection: close
X-Served-By: bitbucket02
Content-Language: en
X-Static-Version: c67fb01467cf
Vary: Accept-Language, Cookie
X-Version: 14f9c66ad9db
```

```
X-Request-Count: 97
Strict-Transport-Security: max-age=2592000
X-Content-Type-Options: nosniff
```

4.4.2 原理分析

我们使用Python中的cookielib模块创建了一个cookie容器cj。登录数据使用urllib.urlencode()方法编码。urllib2模块中有个build_opener()方法，其参数是一个HTTPCookieProcessor类实例。我们要把之前创建的cookie容器传给HTTPCookieProcessor类的构造方法。urllib2.build_opener()方法的返回值是一个URL打开器。我们要调用这个打开器两次：一次访问登录页面，一次访问网站的首页。从响应的首部可以看出，在Set-Cookie首部中只设定了一个cookie，即bb_session。cookielib模块的更多信息可以在Python官方文档中查看，网址是http://docs.python.org/2/library/cookielib.html。

4.5 提交网页表单

浏览网络时，一天之中我们要提交好多次网页表单。现在，我们要使用Python代码提交表单。

4.5.1 准备工作

这个攻略用到了一个Python第三方模块，叫作requests。这个模块的安装方法参见安装指南：http://docs.python-requests.org/en/latest/user/install/。例如，可以在命令行中使用pip安装requests模块，如下所示：

```
$ pip install requests
```

4.5.2 实战演练

让我们来提交一些虚拟数据，注册Twitter账户。提交表单可以使用两种请求方法：GET和POST。不太敏感的数据，例如搜索查询，一般使用GET请求提交。敏感的数据则通过POST请求发送。我们来试一下使用这两种方法提交数据。

代码清单4-4说明了如何提交网页表单，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 4
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import requests
import urllib
```

```
import urllib2

ID_USERNAME = 'signup-user-name'
ID_EMAIL = 'signup-user-email'
ID_PASSWORD = 'signup-user-password'
USERNAME = 'username'
EMAIL = 'you@email.com'
PASSWORD = 'yourpassword'
SIGNUP_URL = 'https://twitter.com/account/create'

def submit_form():
    """Submit a form"""
    payload = {ID_USERNAME : USERNAME,
               ID_EMAIL    :  EMAIL,
               ID_PASSWORD : PASSWORD,}

    # make a get request
    resp = requests.get(SIGNUP_URL)
    print "Response to GET request: %s" %resp.content

    # send POST request
    resp = requests.post(SIGNUP_URL, payload)
    print "Headers from a POST request response: %s" %resp.headers
    #print "HTML Response: %s" %resp.read()

if __name__ == '__main__':
    submit_form()
```

运行这个脚本后，会看到如下输出：

```
$ python 4_4_submit_web_form.py
Response to GET request: <?xml version="1.0" encoding="UTF-8"?>
<hash>
  <error>This method requires a POST.</error>
  <request>/account/create</request>
</hash>

Headers from a POST request response: {'status': '200 OK', 'content-
length': '21064', 'set-cookie': '_twitter_sess=BAh7CD--
d2865d40d1365eeb2175559dc5e6b99f64ea39ff; domain=.twitter.com;
path=/; HttpOnly', 'expires': 'Tue, 31 Mar 1981 05:00:00 GMT',
'vary': 'Accept-Encoding', 'last-modified': 'Sun, 05 May 2013
15:59:27 GMT', 'pragma': 'no-cache', 'date': 'Sun, 05 May 2013
15:59:27 GMT', 'x-xss-protection': '1; mode=block', 'x-transaction':
'a4b425eda23b5312', 'content-encoding': 'gzip', 'strict-transport-
security': 'max-age=631138519', 'server': 'tfe', 'x-mid':
'f7cde9a3f3d111310427116adc90bf3e8c95e868', 'x-runtime': '0.09969',
'etag': '"7af6f92a7f7b4d37a6454caa6094071d"', 'cache-control': 'no-
cache, no-store, must-revalidate, pre-check=0, post-check=0', 'x-
frame-options': 'SAMEORIGIN', 'content-type': 'text/html;
charset=utf-8'}
```

4.5.3 原理分析

这个攻略使用了第三方模块requests。这个模块提供了便利的包装方法get()和post()，能正确编码URL中的数据并提交表单。

在这个攻略中，我们创建了一个数据字典，包含用户名、密码和电子邮件地址，用于注册Twitter账户。我们首先使用GET方法提交表单，但Twitter返回一个错误，说页面只支持POST方法。然后我们使用POST方法提交数据，结果Twitter接受了注册请求，这一点可以由首部数据证实。

4.6 通过代理服务器发送 Web 请求

你可能想通过代理访问网页。如果你为浏览器配置了一个代理服务器，而且代理可用，就可以运行这个攻略。否则，可以使用网上其他可用的公共代理服务器。

4.6.1 准备工作

你需要一个可使用的代理服务器。你可以使用谷歌或其他搜索引擎找到一个免费的代理服务器。这里，为了演示，我们使用的代理服务器是165.24.10.8。

4.6.2 实战演练

我们来通过一个公共代理服务器发送HTTP请求。

代码清单4-5说明了如何通过代理服务器发送Web请求，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 4
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import urllib

URL = 'https://www.github.com'
PROXY_ADDRESS = "165.24.10.8:8080" # By Googling free proxy server

if __name__ == '__main__':
    resp = urllib.urlopen(URL, proxies = {"http" : PROXY_ADDRESS})
    print "Proxy server returns response headers: %s " %resp.headers
```

运行这个脚本后，会看到如下输出：

```
$ python 4_5_proxy_web_request.py
Proxy server returns response headers: Server: GitHub.com
Date: Sun, 05 May 2013 16:16:04 GMT
```

```
Content-Type: text/html; charset=utf-8
Connection: close
Status: 200 OK
Cache-Control: private, max-age=0, must-revalidate
Strict-Transport-Security: max-age=2592000
X-Frame-Options: deny
Set-Cookie: logged_in=no; domain=.github.com; path=/; expires=Thu, 05-
May-2033 16:16:04 GMT; HttpOnly
Set-Cookie: _gh_sess=BAh7...; path=/; expires=Sun, 01-Jan-2023 00:00:00
GMT; secure; HttpOnly
X-Runtime: 8
ETag: "66fcc37865eb05c19b2d15fbb44cd7a9"
Content-Length: 10643
Vary: Accept-Encoding
```

4.6.3　原理分析

这个攻略很简短，使用在谷歌中找到的一个公共代理服务器访问社会化代码分享网站www.github.com。代理服务器的地址传给`urllib`模块的`urlopen()`方法。我们把响应的HTTP首部打印出来，以证明代理设置起到了作用。

4.7　使用 HEAD 请求检查网页是否存在

你可能想在不下载HTML内容的前提下检查网页是否存在。此时我们要使用浏览器客户端发送`get HEAD`请求。根据维基百科中的定义，`HEAD`请求和`GET`请求的响应一样，只是前者没有响应主体。使用`HEAD`请求可以获取响应首部中的元信息，而不用传输整个网页的内容。

4.7.1　实战演练

我们要向www.python.org发送一个`HEAD`请求。这个请求不会下载首页的内容，而是检查服务器是否返回正确的响应，例如`OK`、`FOUND`和`MOVED PERMANENTLY`等。

代码清单4-6说明了如何使用`HEAD`请求检查网页，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 4
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import httplib
import urlparse
import re
import urllib
```

```
DEFAULT_URL = 'http://www.python.org'
HTTP_GOOD_CODES =  [httplib.OK, httplib.FOUND, httplib.MOVED_PERMANENTLY]

def get_server_status_code(url):
    """
    Download just the header of a URL and
    return the server's status code.
    """
    host, path = urlparse.urlparse(url)[1:3]
    try:
        conn = httplib.HTTPConnection(host)
        conn.request('HEAD', path)
        return conn.getresponse().status
    except StandardError:
        return None

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Example HEAD Request')
    parser.add_argument('--url', action="store", dest="url", default=DEFAULT_URL)
    given_args = parser.parse_args()
    url = given_args.url
    if get_server_status_code(url) in HTTP_GOOD_CODES:
        print "Server: %s status is OK: " %url
    else:
        print "Server: %s status is NOT OK!" %url
```

运行这个脚本后，会根据HEAD请求的响应显示成功消息或错误消息，如下所示：

```
$ python 4_6_checking_webpage_with_HEAD_request.py
Server: http://www.python.org status is OK!

$ python 4_6_checking_webpage_with_HEAD_request.py --url=http://www.zytho.org
Server: http://www.zytho.org status is NOT OK!
```

4.7.2 原理分析

我们使用httplib模块中的HTTPConnection()方法向服务器发起HEAD请求。如果需要，可以指定要访问的路径。在这个攻略中，HTTPConnection()方法检查的是www.python.org首页。如果URL不正确，在返回码的可接受列表中就无法找到返回的响应。

4.8 把客户端伪装成 Mozilla Firefox

在Python代码中，你可能想假装成在使用Mozilla Firefox访问Web服务器。

4.8.1 实战演练

你可以在HTTP请求首部中发送自己定制的用户代理值。

代码清单4-7说明了如何把客户端伪装成Mozilla Firefox浏览器，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 4
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import urllib2

BROWSER = 'Mozilla/5.0 (Windows NT 5.1; rv:20.0) Gecko/20100101 Firefox/20.0'
URL = 'http://www.python.org'

def spoof_firefox():
    opener = urllib2.build_opener()
    opener.addheaders = [('User-agent', BROWSER)]
    result = opener.open(URL)
    print "Response headers:"
    for header in  result.headers.headers:
        print "\t",header

if __name__ == '__main__':
    spoof_firefox()
```

运行这个脚本后，会看到如下输出：

```
$ python 4_7_spoof_mozilla_firefox_in_client_code.py
Response headers:
    Date: Sun, 05 May 2013 16:56:36 GMT
    Server: Apache/2.2.16 (Debian)
    Last-Modified: Sun, 05 May 2013 00:51:40 GMT
    ETag: "105800d-5280-4dbedfcb07f00"
    Accept-Ranges: bytes
    Content-Length: 21120
    Vary: Accept-Encoding
    Connection: close
    Content-Type: text/html
```

4.8.2　原理分析

我们使用`urllib2`模块中的`build_opener()`方法创建自定义浏览器，把用户代理字符串设为`Mozilla/5.0 (Windows NT 5.1; rv:20.0) Gecko/20100101 Firefox/20.0`。

4.9　使用 HTTP 压缩节省 Web 请求消耗的带宽

你可能想让Web服务器在下载网页时有更好的性能表现。压缩HTTP数据能提升伺服网页内容的速度。

4.9.1　实战演练

我们来编写一个Web服务器，把内容压缩成gzip格式后再提供给访问者。

代码清单4-8说明了如何压缩HTTP数据，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 4
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import string
import os
import sys
import gzip
import cStringIO
from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer

DEFAULT_HOST = '127.0.0.1'
DEFAULT_PORT = 8800
HTML_CONTENT = """<html><body><h1>Compressed Hello  World!</h1></body></html>"""

class RequestHandler(BaseHTTPRequestHandler):
    """ Custom request handler"""

    def do_GET(self):
        """ Handler for the GET requests """
        self.send_response(200)
        self.send_header('Content-type','text/html')
        self.send_header('Content-Encoding','gzip')
        zbuf = self.compress_buffer(HTML_CONTENT)
        sys.stdout.write("Content-Encoding: gzip\r\n")
        self.send_header('Content-Length',len(zbuf))
        self.end_headers()

        # Send the message to browser
        zbuf = self.compress_buffer(HTML_CONTENT)
        sys.stdout.write("Content-Encoding: gzip\r\n")
        sys.stdout.write("Content-Length: %d\r\n" % (len(zbuf)))
        sys.stdout.write("\r\n")
        self.wfile.write(zbuf)
        return

    def compress_buffer(self, buf):
        zbuf = cStringIO.StringIO()
        zfile = gzip.GzipFile(mode = 'wb',  fileobj = zbuf, compresslevel = 6)
        zfile.write(buf)
        zfile.close()
        return zbuf.getvalue()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Simple HTTP Server Example')
```

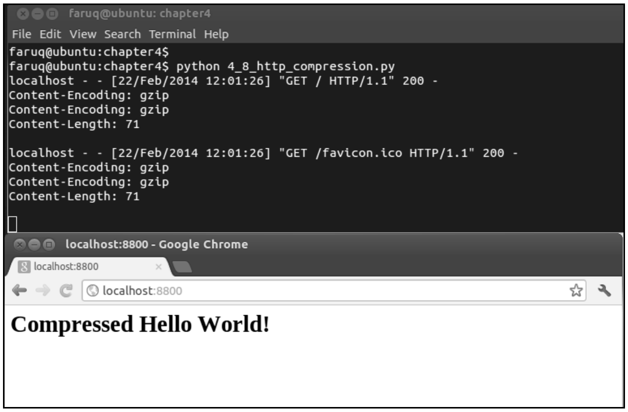
4

```
    parser.add_argument('--port', action="store", dest="port", type=int,
default=DEFAULT_PORT)
    given_args = parser.parse_args()
    port = given_args.port
    server_address =  (DEFAULT_HOST, port)
    server = HTTPServer(server_address, RequestHandler)
    server.serve_forever()
```

运行这个脚本后，在浏览器中访问http://localhost:8800，会看到浏览器中显示了文本“Compressed Hello World!”（HTTP压缩后得到的结果），如下所示：

```
$ python 4_8_http_compression.py
localhost - - [22/Feb/2014 12:01:26] "GET / HTTP/1.1" 200 -
Content-Encoding: gzip
Content-Encoding: gzip
Content-Length: 71
localhost - - [22/Feb/2014 12:01:26] "GET /favicon.ico HTTP/1.1" 200 -
Content-Encoding: gzip
Content-Encoding: gzip
Content-Length: 71
```

下面的截图展示了Web服务器伺服压缩内容的过程：



4.9.2 原理分析

我们实例化`BaseHTTPServer`模块中的`HTTPServer`类，创建了一个Web服务器。然后为这个服务器实例定义了一个请求处理方法，它使用`compress_buffer()`方法压缩发给客户端的每个响应，再把事先定义好的HTML内容发送给客户端。

4.10 编写一个支持断点续传功能的 HTTP 容错客户端

你可能想编写一个容错的客户端，它在因某种原因初次下载失败后能继续下载文件。

4.10.1　实战演练

让我们从www.python.org上下载Python 2.7的源码。使用`resume_download()`函数下载的文件在中止后能继续下载尚未下载的内容。

代码清单4-9说明了如何继续下载，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 4
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import urllib
import os

TARGET_URL = 'http://python.org/ftp/python/2.7.4/'
TARGET_FILE = 'Python-2.7.4.tgz'

class CustomURLOpener(urllib.FancyURLopener):
    """Override FancyURLopener to skip error 206 (when a
       partial file is being sent)
    """
    def http_error_206(self, url, fp, errcode, errmsg, headers, data=None):
        pass

def resume_download():
    file_exists = False
    CustomURLClass = CustomURLOpener()
    if os.path.exists(TARGET_FILE):
        out_file = open(TARGET_FILE,"ab")
        file_exists = os.path.getsize(TARGET_FILE)
        #If the file exists, then only download the unfinished part
        CustomURLClass.addheader("range","bytes=%s-" % (file_exists))
    else:
        out_file = open(TARGET_FILE,"wb")

    web_page = CustomURLClass.open(TARGET_URL + TARGET_FILE)

    #Check if last download was OK
    if int(web_page.headers['Content-Length']) == file_exists:
        loop = 0
        print "File already downloaded!"

    byte_count = 0
    while True:
        data = web_page.read(8192)
        if not data:
            break
        out_file.write(data)
        byte_count = byte_count + len(data)

    web_page.close()
```

```
    out_file.close()

    for k,v in web_page.headers.items():
        print k, "=",v
    print "File copied", byte_count, "bytes from", web_page.url

if __name__ == '__main__':
    resume_download()
```

运行这个脚本后，会看到如下输出结果：

```
$ python 4_9_http_fail_over_client.py
content-length = 14489063
content-encoding = x-gzip
accept-ranges = bytes
connection = close
server = Apache/2.2.16 (Debian)
last-modified = Sat, 06 Apr 2013 14:16:10 GMT
content-range = bytes 0-14489062/14489063
etag = "1748016-dd15e7-4d9b1d8685e80"
date = Tue, 07 May 2013 12:51:31 GMT
content-type = application/x-tar
File copied 14489063 bytes from http://python.org/ftp/python/2.7.4/Python-2.7.4.tgz
```

4.10.2　原理分析

在这个攻略中，我们定义了一个URL打开器类，继承自`urllib`模块中的`FancyURLopener`类，不过重定义了用于分段下载内容的`http_error_206()`方法。`resume_download()`函数首先检查目标文件是否存在，如果不存在就尝试使用自定义的URL打开器类下载。

4.11　使用 Python 和 OpenSSL 编写一个简单的 HTTPS 服务器

你需要使用Python编写一个安全的Web服务器，而且已经有了SSL密钥和证书文件。

4.11.1　准备工作

你需要安装第三方Python模块`pyOpenSSL`。这个模块可从PyPI上下载，地址为https://pypi.python.org/pypi/pyOpenSSL。在Windows和Linux主机上都要安装一些其他的包，在http://pythonhosted.org/pyOpenSSL/中有说明。

4.11.2　实战演练

把证书文件放在当前工作目录后，我们就可以创建一个Web服务器，利用这个证书向客户端发送加密后的内容。

代码清单4-10是安全HTTP服务器的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 4
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.
# Requires pyOpenSSL and SSL packages installed

import socket, os
from SocketServer import BaseServer
from BaseHTTPServer import HTTPServer
from SimpleHTTPServer import SimpleHTTPRequestHandler
from OpenSSL import SSL

class SecureHTTPServer(HTTPServer):
    def __init__(self, server_address, HandlerClass):
        BaseServer.__init__(self, server_address, HandlerClass)
        ctx = SSL.Context(SSL.SSLv23_METHOD)
        fpem = 'server.pem' # location of the server private key and the server
certificate
        ctx.use_privatekey_file (fpem)
        ctx.use_certificate_file(fpem)
        self.socket = SSL.Connection(ctx, socket.socket(self.address_family,
                                                        self.socket_type))
        self.server_bind()
        self.server_activate()

class SecureHTTPRequestHandler(SimpleHTTPRequestHandler):
    def setup(self):
        self.connection = self.request
        self.rfile = socket._fileobject(self.request, "rb", self.rbufsize)
        self.wfile = socket._fileobject(self.request, "wb", self.wbufsize)

def run_server(HandlerClass = SecureHTTPRequestHandler,
        ServerClass = SecureHTTPServer):
    server_address = ('', 4443) # port needs to be accessible by user
    server = ServerClass(server_address, HandlerClass)
    running_address = server.socket.getsockname()
    print "Serving HTTPS Server on %s:%s ..." %(running_address[0],
running_address[1])
    server.serve_forever()

if __name__ == '__main__':
    run_server()
```

运行这个脚本后会看到如下输出：

```
$ python 4_10_https_server.py
Serving HTTPS Server on 0.0.0.0:4443 ...
```

4

4.11.3　原理分析

如果你仔细观察前面编写Web服务器的攻略会发现，它和这个攻略没有太多的区别，基本流程都是一样的。最重要的一个不同点是，这个脚本调用了`SSL.Context()`方法，并将其参数设为`SSL.SSLv23_METHOD`。我们使用Python OpenSSL第三方模块提供的`Connection`类创建了一个SSL套接字。`Connection`类构造方法的参数是前面创建的上下文对象，以及地址族和套接字类型。

服务器的证书文件保存在当前目录中。证书文件提供给上下文对象使用。最后，调用`server_activate()`方法激活服务器。

第 5 章

电子邮件协议、FTP和CGI编程

本章攻略：

- ❑ 列出远程FTP服务器中的文件
- ❑ 把本地文件上传到远程FTP服务器中
- ❑ 把当前工作目录中的内容压缩成ZIP文件后通过电子邮件发送
- ❑ 通过POP3协议下载谷歌电子邮件
- ❑ 通过IMAP协议查收远程服务器中的电子邮件
- ❑ 通过Gmail的SMTP服务器发送带有附件的电子邮件
- ❑ 使用CGI为基于Python的Web服务器编写一个留言板

5.1　简介

本章通过Python攻略介绍FTP、电子邮件和CGI通信协议。Python这门语言很高效也很友好。使用Python可以很方便地实现简单的FTP操作，例如下载和上传文件。

本章有些有趣的攻略，例如在Python脚本中管理谷歌电子邮件（也叫Gmail）。使用这些攻略可以通过IMAP、POP3和SMTP协议查收、下载和发送电子邮件。还有一个攻略编写了一个支持CGI功能的Web服务器，演示了基本的CGI操作，例如编写Web应用中的游客留言表单。

5.2　列出 FTP 远程服务器中的文件

你可能想列出Linux内核FTP网站（ftp.kernel.org）中的文件。这个攻略也可用在其他FTP网站上。

5.2.1　准备工作

如果处理需要账户的FTP网站，你需要提供用户名和密码。但在这个攻略中不需要Linux内核

FTP网站的用户名和密码，因为可以匿名登录。

5.2.2 实战演练

要从选中的FTP网站中获取文件，可以使用ftplib库。这个库的详细文档可在http://docs.python.org/2/library/ftplib.html中查看。

我们来看一下如何使用ftplib获取文件。

代码清单5-1是一次简单的FTP连接测试，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 5
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

FTP_SERVER_URL = 'ftp.kernel.org'

import ftplib

def test_ftp_connection(path, username, email):
    #Open ftp connection
    ftp = ftplib.FTP(path, username, email)

    #List the files in the /pub directory
    ftp.cwd("/pub")
    print "File list at %s:" %path
    files = ftp.dir()
    print files

    ftp.quit()

if __name__ == '__main__':
    test_ftp_connection(path=FTP_SERVER_URL, username='anonymous',
                        email='nobody@nourl.com')
```

这个攻略会列出FTP路径（ftp.kernel.org/pub）中的文件和文件夹。运行这个脚本后，会看到如下输出：

```
$ python 5_1_list_files_on_ftp_server.py
File list at ftp.kernel.org:
drwxrwxr-x    6 ftp      ftp          4096 Dec 01  2011 dist
drwxr-xr-x   13 ftp      ftp          4096 Nov 16  2011 linux
drwxrwxr-x    3 ftp      ftp          4096 Sep 23  2008 media
drwxr-xr-x   17 ftp      ftp          4096 Jun 06  2012 scm
drwxrwxr-x    2 ftp      ftp          4096 Dec 01  2011 site
drwxr-xr-x   13 ftp      ftp          4096 Nov 27  2011 software
drwxr-xr-x    3 ftp      ftp          4096 Apr 30  2008 tools
```

5.2.3 原理分析

这个攻略使用ftplib创建了一个连接到ftp.kernel.org/pub上的FTP客户端会话。test_ftp_connection()函数的参数是连接FTP服务器所需的路径、用户名和电子邮件地址。

FTP客户端会话使用ftplib库中的FTP()函数创建，其参数就是传入test_ftp_connection()的参数。FTP()函数返回一个客户端句柄，用于运行常用的FTP命令，例如切换工作目录的命令cwd()。dir()方法返回目录的文件夹结构。

处理完成后，最后调用ftp.quit()终止FTP会话。

5.3 把本地文件上传到远程 FTP 服务器中

你可能想把文件上传到FTP服务器中。

5.3.1 准备工作

让我们来搭建一个本地FTP服务器。在Unix/Linux中，可以使用下述命令安装wu-ftpd包：

```
$ sudo apt-get install wu-ftpd
```


在运行Windows的设备中，可以安装FileZilla FTP服务器，下载地址为https://filezilla-project.org/download.php?type=server。

你应该按照FTP服务器包的说明来创建一个FTP账户。

你可能还想上传一个文件到FTP服务器中。你可以指定服务器地址、登录凭据和文件名作为脚本的输入参数。你应该在本地新建一个文件，命名为readme.txt，在里面随便写些文本。

5.3.2 实战演练

我们使用下面的脚本来搭建一个FTP本地服务器。在Unix/Linux中，可以安装wu-ftpd包。然后，可以把文件上传到已登录用户的家目录中。你可以指定服务器地址、登录凭据和文件名作为脚本的输入参数。

代码清单5-2是FTP上传文件示例，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 5
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.
```

```
import os
import argparse
import ftplib
import getpass

LOCAL_FTP_SERVER = 'localhost'
LOCAL_FILE = 'readme.txt'

def ftp_upload(ftp_server, username, password, file_name):
    print "Connecting to FTP server: %s" %ftp_server
    ftp = ftplib.FTP(ftp_server)
    print "Login to FTP server: user=%s" %username
    ftp.login(username, password)
    ext = os.path.splitext(file_name)[1]
    if ext in (".txt", ".htm", ".html"):
        ftp.storlines("STOR " + file_name, open(file_name))
    else:
        ftp.storbinary("STOR " + file_name, open(file_name, "rb"), 1024)
    print "Uploaded file: %s" %file_name

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='FTP Server Upload Example')
    parser.add_argument('--ftp-server', action="store", dest="ftp_server",
default=LOCAL_FTP_SERVER)
    parser.add_argument('--file-name', action="store", dest="file_name",
default=LOCAL_FILE)
    parser.add_argument('--username', action="store", dest="username",
default=getpass.getuser())
    given_args = parser.parse_args()
    ftp_server, file_name, username = given_args.ftp_server, given_args.file_name,
given_args.username
    password = getpass.getpass(prompt="Enter you FTP password: ")
    ftp_upload(ftp_server, username, password, file_name)
```

如果搭建了本地FTP服务器，运行下面这个脚本后会登入FTP服务器并上传文件。如果命令行中没有指定默认文件名，这个脚本将上传readme.txt文件。

```
$ python 5_2_upload_file_to_ftp_server.py
Enter your FTP password:
Connecting to FTP server: localhost
Login to FTP server: user=faruq
Uploaded file: readme.txt

$ cat /home/faruq/readme.txt
This file describes what to do with the .bz2 files you see elsewhere
on this site (ftp.kernel.org).
```

5.3.3　原理分析

在这个攻略中，我们假设本地FTP服务器正在运行中。除此之外还可以连接远程FTP服务器。在`ftp_upload()`方法中，使用Python中的`ftplib`模块提供的`FTP()`函数创建一个FTP连接对象。

然后调用login()方法，登录服务器。

登录成功后，在ftp对象上调用方法storlines()或storbinary()执行STOR命令。前一个方法用于发送ASCII文本文件，例如HTML或纯文本文件。后一个方法用于发送二进制数据，例如压缩文件。

这些FTP方法最好放在try-catch错误处理块中，为了行文简洁，这里并没有这么做。

5.4　把当前工作目录中的内容压缩成 ZIP 文件后通过电子邮件发送

如果能把当前工作目录中的内容压缩成ZIP文件发送给别人，该多么有趣啊。使用这个攻略，你可以快速和朋友共享文件。

5.4.1　准备工作

如果你的设备中没有安装任何邮件服务器，你需要安装一个本地邮件服务器，例如postfix。在Debian/Ubuntu系统中可以使用apt-get的默认设置安装，如下面的命令所示：

```
$ sudo apt-get install postfix
```

5

5.4.2　实战演练

我们首先要压缩当前目录，然后创建一封电子邮件。电子邮件可通过外部的SMTP主机发送，也可使用本地电子邮件服务器发送。和其他的攻略类似，发件人和收件人信息都从命令行参数中获取。

代码清单5-3展示了如何把文件夹压缩成ZIP文件再通过电子邮件发送，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 5
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import os
import argparse
import smtplib
import zipfile
import tempfile
from email import encoders
from email.mime.base import MIMEBase
from email.mime.multipart import MIMEMultipart

def email_dir_zipped(sender, recipient):
    zf = tempfile.TemporaryFile(prefix='mail', suffix='.zip')
```

```
    zip = zipfile.ZipFile(zf, 'w')
    print "Zipping current dir: %s" %os.getcwd()
    for file_name in os.listdir(os.getcwd()):
        zip.write(file_name)
    zip.close()
    zf.seek(0)

    # Create the message
    print "Creating email message..."
    email_msg = MIMEMultipart()
    email_msg['Subject'] = 'File from path %s' %os.getcwd()
    email_msg['To'] = ', '.join(recipient)
    email_msg['From'] = sender
    email_msg.preamble = 'Testing email from Python.\n'
    msg = MIMEBase('application', 'zip')
    msg.set_payload(zf.read())
    encoders.encode_base64(msg)
    msg.add_header('Content-Disposition', 'attachment',
                   filename=os.getcwd()[-1] + '.zip')
    email_msg.attach(msg)
    email_msg = email_msg.as_string()

    # send the message
    print "Sending email message..."
    try:
        smtp = smtplib.SMTP('localhost')
        smtp.set_debuglevel(1)
        smtp.sendmail(sender, recipient, email_msg)
    except Exception, e:
        print "Error: %s" %str(e)
    finally:
        smtp.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Email Example')
    parser.add_argument('--sender', action="store", dest="sender",
default='you@you.com')
    parser.add_argument('--recipient', action="store", dest="recipient")
    given_args = parser.parse_args()
    email_dir_zipped(given_args.sender, given_args.recipient)
```

运行这个脚本后看到的输出如下所示。因为开启了电子邮件调试模式，所以还显示了一些额外信息。

```
$ python 5_3_email_current_dir_zipped.py --recipient=faruq@localhost
Zipping current dir: /home/faruq/Dropbox/PacktPub/pynet-cookbook/
pynetcookbook_code/chapter5
Creating email message...
Sending email message...
send: 'ehlo [127.0.0.1]\r\n'
reply: '250-debian6.debian2013.com\r\n'
reply: '250-PIPELINING\r\n'
reply: '250-SIZE 10240000\r\n'
```

```
reply: '250-VRFY\r\n'
reply: '250-ETRN\r\n'
reply: '250-STARTTLS\r\n'
reply: '250-ENHANCEDSTATUSCODES\r\n'
reply: '250-8BITMIME\r\n'
reply: '250 DSN\r\n'
reply: retcode (250); Msg: debian6.debian2013.com
PIPELINING
SIZE 10240000
VRFY
ETRN
STARTTLS
ENHANCEDSTATUSCODES
8BITMIME
DSN
send: 'mail FROM:<you@you.com> size=9141\r\n'
reply: '250 2.1.0 Ok\r\n'
reply: retcode (250); Msg: 2.1.0 Ok
send: 'rcpt TO:<faruq@localhost>\r\n'
reply: '250 2.1.5 Ok\r\n'
reply: retcode (250); Msg: 2.1.5 Ok
send: 'data\r\n'
reply: '354 End data with <CR><LF>.<CR><LF>\r\n'
reply: retcode (354); Msg: End data with <CR><LF>.<CR><LF>
data: (354, 'End data with <CR><LF>.<CR><LF>')
send: 'Content-Type: multipart/mixed;
boundary="===============0388489101==...[TRUNCATED]
reply: '250 2.0.0 Ok: queued as 42D2F34A996\r\n'
reply: retcode (250); Msg: 2.0.0 Ok: queued as 42D2F34A996
data: (250, '2.0.0 Ok: queued as 42D2F34A996')
```

5.4.3　原理分析

为了通过电子邮件发送压缩后的文件夹，我们用到了Python中的zipfile、smtplib和email三个模块。这在email_dir_zipped()方法中实现。这个方法接收两个参数：发件人和收件人的电子邮件地址。

为了压缩文件，我们使用tempfile模块中的TemporaryFile类新建了一个临时文件，把这个文件的前缀设为mail，后缀设为.zip。然后把临时文件作为参数传给ZipFile类的构造方法，初始化一个ZIP压缩文件对象。接着在这个压缩对象上调用write()方法，添加当前目录中的文件。

为了发送电子邮件，我们使用email.mime.multipart模块中的MIMEmultipart()类创建了一个MIME为multipart的邮件。和普通的电子邮件一样，主题、收件人和发件人信息都在电子邮件的首部中设定。

电子邮件的附件使用MIMEBase()方法创建。我们首先设置application/zip首部，然后在附件对象上调用set_payload()方法。为了正确地编码消息，调用了encoders模块中的

encode_base64()方法。add_header()方法能帮助我们设定附件的首部。至此，附件已经准备好，可以添加到邮件中了，因此我们调用了attach()方法。

若想发送电子邮件，要调用smtplib模块中的SMTP类创建一个实例。在这个实例上调用sendmail()方法后，会利用操作系统中的程序正确地把电子邮件发送出去。发送的细节被隐藏了，不过，如果你想看到详细的过程，可以打开调试模式，如前所示。

5.4.4　参考资源

- 本节用到的Python库详情请参阅文档，地址为http://docs.python.org/2/library/smtplib.html。

5.5　通过 POP3 协议下载谷歌电子邮件

你可能想通过POP3协议下载谷歌（或者其他任何一个电子邮件服务提供商）账户中的电子邮件。

5.5.1　准备工作

要运行这个攻略，你需要有谷歌或其他服务提供商的电子邮件账户。

5.5.2　实战演练

我们要尝试下载用户谷歌电子邮件账户中的第一封邮件。用户名在命令行中输入，但为了保密，密码不能在命令行中指定，而是在运行脚本时输入，而且不能显示出来。

代码清单5-4展示了如何通过POP3协议下载谷歌账户中的电子邮件，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 5
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import getpass
import poplib

GOOGLE_POP3_SERVER = 'pop.googlemail.com'

def download_email(username):
    mailbox = poplib.POP3_SSL(GOOGLE_POP3_SERVER, '995')
    mailbox.user(username)
    password = getpass.getpass(prompt="Enter your 谷歌 password: ")
    mailbox.pass_(password)
```

```
    num_messages = len(mailbox.list()[1])
    print "Total emails: %s" %num_messages
    print "Getting last message"
    for msg in mailbox.retr(num_messages)[1]:
        print msg
    mailbox.quit()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Email Download Example')
    parser.add_argument('--username', action="store", dest="username",
default=getpass.getuser())
    given_args = parser.parse_args()
    username = given_args.username
    download_email(username)
```

运行这个脚本后会看到类似下面的输出。为了保护隐私，输出的内容有所删减。

```
$ python 5_4_download_google_email_via_pop3.py --username=<USERNAME>
Enter your 谷歌 password:
Total emails: 333
Getting last message
...[TRUNCATED]
```

5.5.3　原理分析

这个攻略通过POP3协议从用户的谷歌账户中下载第一封邮件。在`download_email()`函数中，使用`poplib`模块中的`POP3_SSL`类创建了一个`mailbox`对象。在`POP3_SSL`类的构造方法中，我们传入了谷歌POP3服务器的地址和端口号。然后在`mailbox`对象上调用`user()`方法，设定用户的账户。密码使用`getpass`模块中的`getpass()`方法以一种安全的方式从用户的输入中获取，然后传给`mailbox`对象。在`mailbox`对象上调用`list()`方法会以一个Python列表的形式返回电子邮件。

这个脚本先显示电子邮件的数量，然后调用`retr()`方法取回第一封邮件。最后，在`mailbox`对象上调用`quit()`方法，安全地结束连接。

5.6　通过 IMAP 协议查收远程服务器中的电子邮件

除了使用POP3协议之外，还可以使用IMAP协议从谷歌账户中取回电子邮件。使用这种方法，取回后邮件不会被删除。

5.6.1　准备工作

要运行这个攻略，你需要有谷歌或其他服务提供商的电子邮件账户。

5.6.2　实战演练

让我们连接到谷歌的电子邮件账户，读取第一封电子邮件。如果你没有删除，第一封电子邮件应该是来自谷歌的欢迎消息。

代码清单5-5展示了如何通过IMAP协议查收谷歌账户中的电子邮件，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 5
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import getpass
import imaplib

GOOGLE_IMAP_SERVER = 'imap.googlemail.com'

def check_email(username):
    mailbox = imaplib.IMAP4_SSL(GOOGLE_IMAP_SERVER, '993')
    password = getpass.getpass(prompt="Enter your 谷歌 password: ")
    mailbox.login(username, password)
    mailbox.select('Inbox')
    typ, data = mailbox.search(None, 'ALL')
    for num in data[0].split():
        typ, data = mailbox.fetch(num, '(RFC822)')
        print 'Message %s\n%s\n' % (num, data[0][1])
        break
    mailbox.close()
    mailbox.logout()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Email Download Example')
    parser.add_argument('--username', action="store", dest="username",
default=getpass.getuser())
    given_args = parser.parse_args()
    username = given_args.username
    check_email(username)
```

运行这个脚本后会看到如下输出。为了保护隐私，删减了一些信息。

```
$ python 5_5_check_remote_email_via_imap.py --username=<USER_NAME>
Enter your 谷歌 password:
Message 1
Received: by 10.140.142.16; Sat, 17 Nov 2007 09:26:31 -0800 (PST)
Message-ID: <...>@mail.gmail.com>
Date: Sat, 17 Nov 2007 09:26:31 -0800
From: "Gmail Team" <mail-noreply@google.com>
To: "<User Full Name>" <USER_NAME>@gmail.com>
Subject: Gmail is different. Here's what you need to know.
MIME-Version: 1.0
Content-Type: multipart/alternative;
```

```
    boundary="----=_Part_7453_30339499.1195320391988"
------=_Part_7453_30339499.1195320391988
Content-Type: text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: 7bit
Content-Disposition: inline

Messages that are easy to find, an inbox that organizes itself, great
spam-fighting tools and built-in chat. Sound cool? Welcome to Gmail.
To get started, you may want to:
[TRUNCATED]
```

5.6.3 原理分析

上述脚本从命令行参数中获取谷歌用户名，然后调用`check_email()`函数。在这个函数中，使用`imaplib`模块中的`IMAP4_SSL`类创建了一个`mailbox`对象，实例化时传入了谷歌的IMAP服务器地址和默认的端口号。

然后，在这个函数中使用密码登录账户。密码使用`getpass`模块中的`getpass()`方法从用户输入中捕获。然后在`mailbox`对象上调用`select()`方法，选择收件箱。

在`mailbox`对象上可以调用很多有用的方法，其中两个是`search()`和`fetch()`，在这个脚本中用来获取第一封邮件。最后，在`mailbox`对象上调用`close()`和`logout()`方法，安全地结束IMAP连接。

5

5.7 通过 Gmail 的 SMTP 服务器发送带有附件的电子邮件

你也许想从谷歌的电子邮件账户中发送一封邮件到其他账户中，或许还想为这封邮件附加一个文件。

5.7.1 准备工作

要运行这个攻略，你需要有谷歌或其他服务提供商的电子邮件账户。

5.7.2 实战演练

我们可以创建一封邮件，把Python的LOGO python-logo.gif附加到邮件中，然后从谷歌账户中发给另一个账户。

代码清单5-6展示了如何从谷歌账户中发送电子邮件：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 5
# This program is optimized for Python 2.7.
```

```
# It may run on any other version with/without modifications.

import argparse
import os
import getpass
import re
import sys
import smtplib

from email.mime.image import MIMEImage
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

SMTP_SERVER = 'smtp.gmail.com'
SMTP_PORT = 587

def send_email(sender, recipient):
    """ Send email message """
    msg = MIMEMultipart()
    msg['Subject'] = 'Python Emaill Test'
    msg['To'] = recipient
    msg['From'] = sender
    subject = 'Python email Test'
    message = 'Images attached.'
    # attach imgae files
    files = os.listdir(os.getcwd())
    gifsearch = re.compile(".gif", re.IGNORECASE)
    files = filter(gifsearch.search, files)
    for filename in files:
        path = os.path.join(os.getcwd(), filename)
        if not os.path.isfile(path):
            continue
        img = MIMEImage(open(path, 'rb').read(), _subtype="gif")
        img.add_header('Content-Disposition', 'attachment', filename=filename)
        msg.attach(img)

    part = MIMEText('text', "plain")
    part.set_payload(message)
    msg.attach(part)

    # create smtp session
    session = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
    session.ehlo()
    session.starttls()
    session.ehlo
    password = getpass.getpass(prompt="Enter your 谷歌 password: ")
    session.login(sender, password)
    session.sendmail(sender, recipient, msg.as_string())
    print "Email sent."
    session.quit()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Email Sending Example')
    parser.add_argument('--sender', action="store", dest="sender")
```

```
    parser.add_argument('--recipient', action="store", dest="recipient")
    given_args = parser.parse_args()
    send_email(given_args.sender, given_args.recipient)
```

如果提供的谷歌账户信息正确，运行这个脚本后会看到成功消息，提示把一封邮件发送给了另一个电子邮件地址。运行这个脚本之后，可以到收件人的电子邮件账户中确认是否真的成功发送了邮件。

```
$ python 5_6_send_email_from_gmail.py --sender=<USERNAME>@gmail.com –
recipient=<USER>@<ANOTHER_COMPANY.com>
Enter you Google password:
Email sent.
```

5.7.3　原理分析

在这个攻略的`send_email()`函数中创建了一封邮件。我们为`send_email()`函数提供了谷歌账户，邮件从这个账户中发出。邮件头对象`msg`使用`MIMEMultipart()`方法创建，然后添加了主题、收件人和发件人。

在当前路径中寻找.gif格式图片时用到了Python中的正则表达式处理模块。图片附件对象`img`由`email.mime.image`模块中的`MIMEImage()`方法创建。然后为图片对象设定了正确的首部，最后再把图片附加到前面创建的`msg`对象上。如这个攻略所示，我们可以在`for`循环中附加多个图片。使用类似的方式，还可以添加纯文本附件。

为了发送电子邮件，我们创建了一个SMTP会话，并在这个会话对象上调用了测试方法，例如`ehlo()`和`starttls()`。然后使用用户名和密码登录谷歌的SMTP服务器，再调用`sendmail()`方法发送电子邮件。

5.8　使用 CGI 为基于 Python 的 Web 服务器编写一个留言板

通用网关接口（Common Gateway Interface，简称CGI）是Web编程的一种标准。通过CGI，可以使用脚本生成服务器输出。你可能想获取用户在浏览器中输入的表单数据，重定向到另一个页面，确认收到了用户执行的操作。

5.8.1　实战演练

我们首先要运行一个支持CGI脚本的Web服务器。我们把用Python编写的CGI脚本放在cgi-bin/子目录中，然后访问包含反馈表单的HTML页面。提交表单后，Web服务器把表单数据发送给CGI脚本，我们会看到这个脚本生成的输出。

代码清单5-7展示了如何让使用Python编写的Web服务器支持CGI：

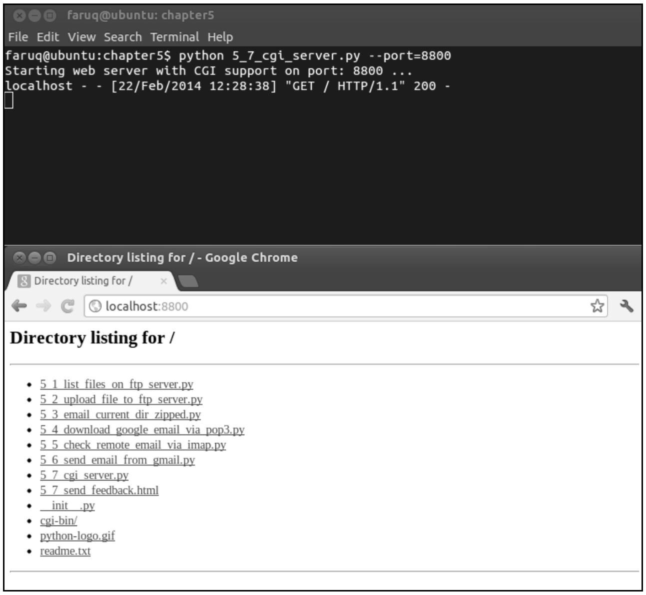
```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 5
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import os
import cgi
import argparse
import BaseHTTPServer
import CGIHTTPServer
import cgitb
cgitb.enable()  ## enable CGI error reporting

def web_server(port):
    server = BaseHTTPServer.HTTPServer
    handler = CGIHTTPServer.CGIHTTPRequestHandler #RequestsHandler
    server_address = ("", port)
    handler.cgi_directories = ["/cgi-bin", ]
    httpd = server(server_address, handler)
    print "Starting web server with CGI support on port: %s ..." %port
    httpd.serve_forever()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='CGI Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int,
required=True)
    given_args = parser.parse_args()
    web_server(given_args.port)
```

下面的截图展示了启用CGI的Web服务器正在伺服内容：



运行这个脚本后，会看到如下输出：

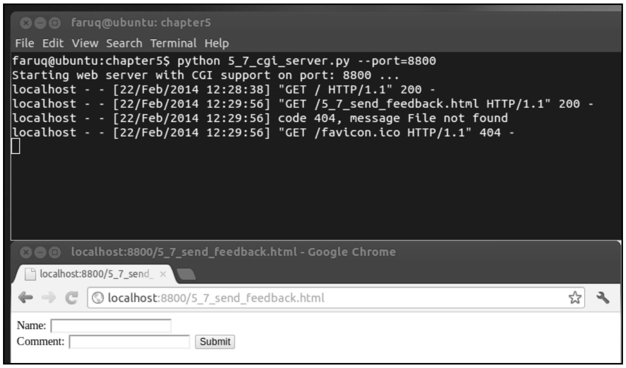
```
$ python 5_7_cgi_server.py --port=8800
Starting web server with CGI support on port: 8800 ...
localhost - - [19/May/2013 18:40:22] "GET / HTTP/1.1" 200 -
```

现在，你要在浏览器中访问http://localhost:8800/5_7_send_feedback.html。

你会看到一个输入表单。假设你在表单中填写了下面的内容：

```
Name: User1
Comment: Comment1
```

下面的截图展示了在网页表单中输入用户评论的过程：



然后浏览器会被重定向到http://localhost:8800/cgi-bin/5_7_get_feedback.py，在这个页面中你会看到如下输出：

```
User1 sends a comment: Comment1
```

用户的评论会显示在浏览器中：



5.8.2 原理分析

我们创建了一个简单的HTTP服务器，以支持处理CGI请求。Python在模块`BaseHTTPServer`

和CGIHTTPserver中提供了这些接口。

我们配置了处理程序，使用路径/cgi-bin存放CGI脚本。其他路径不能运行CGI脚本。

文件5_7_send_feedback.html中的HTML反馈表单显示了一个很简单的HTML表单，包含以下代码：

```
<html>
    <body>
        <form action="/cgi-bin/5_7_get_feedback.py" method="post">
            Name: <input type="text" name="Name">  <br />
            Comment: <input type="text" name="Comment" />
            <input type="submit" value="Submit" />
        </form>
    </body>
</html>
```

注意，这个表单的方法是POST，action属性被设为了/cgi-bin/5_7_get_feedback.py文件。这个文件的内容如下：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 5
# This program requires Python 2.7 or any later version

# Import modules for CGI handling
import cgi
import cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
name = form.getvalue('Name')
comment  = form.getvalue('Comment')

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>CGI Program Example </title>"
print "</head>"
print "<body>"
print "<h2> %s sends a comment: %s</h2>" % (name, comment)
print "</body>"
print "</html>"
```

在这个CGI脚本中，调用cgilib模块中的FieldStorage()方法，得到一个用于处理HTML表单输入的表单对象。然后使用getvalue()方法处理两个输入（name和comment）。最后，这个脚本显示一行文字，提示某个用户发送了一篇评论，作为用户输入的反馈。

第 6 章

屏幕抓取和其他实用程序

本章攻略：

- 使用谷歌地图API搜索公司地址
- 使用谷歌地图URL搜索地理坐标
- 搜索维基百科中的文章
- 使用谷歌搜索股价
- 搜索GitHub中的源代码仓库
- 读取BBC的新闻订阅源
- 爬取网页中的链接

6.1 简介

本章介绍一些有趣的Python脚本，可让你从网络中获取有用信息，例如公司地址、某公司的股价或通讯社网站中的最新资讯。这些Python脚本演示了不使用复杂的API时，如何以更简单的方式获取简要信息。

按照这些攻略的做法，你能编写适应复杂需求的代码，例如，找到一家公司的详细信息，包括所在地、新闻、股价等。

6.2 使用谷歌地图 API 搜索公司地址

你可能想搜索所在地区内一家知名公司的地址。

6.2.1 准备工作

你可以使用Python的地理编码库`pygeocoder`搜索本地公司。你需要使用`pip`或`easy_install`从PyPI上安装这个库，输入的命令为`$ pip install pygeocoder`或`$ easy_install pygeocoder`。

6.2.2　实战演练

我们来使用几行Python代码查找英国著名零售商Argos有限公司的地址。

代码清单6-1是一个简单的地理编码示例，用于搜索一家公司的地址，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 6
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

from pygeocoder import Geocoder

def search_business(business_name):

    results = Geocoder.geocode(business_name)

    for result in results:
        print result

if __name__ == '__main__':
    business_name =  "Argos Ltd, London"
    print "Searching %s" %business_name
    search_business(business_name)
```

这个脚本会打印出Argos有限公司的地址，如下所示。输出的内容根据所安装的地理编码库会有细微的差别。

```
$ python 6_1_search_business_addr.py
Searching Argos Ltd, London

Argos Ltd, 110-114 King Street, London, Greater London W6 0QP, UK
```

6.2.3　原理分析

这个攻略依赖于Python的第三方地理编码库。

这个攻略定义了一个简单的函数，`search_business()`，其参数是公司名，然后它把公司名传给`geocode()`方法。`geocode()`方法可能返回零个或多个结果，具体取决于搜索的关键字。

在这个攻略中，传给`geocode()`方法的搜索关键字是“Argos Ltd, London”。得到的结果是Argos有限公司的地址，即“110-114 King Street, London, Greater London W6 0QP, UK”。

6.2.4　参考资源

`pygeocoder`库很强大，有很多与地理编码有关的有趣和有用的功能。在开发者的网站中有更详细的说明，地址为https://bitbucket.org/xster/pygeocoder/wiki/Home。

6.3　使用谷歌地图 URL 搜索地理坐标

有时你需要一个简单的函数，只通过城市名即可找出城市的地理坐标。你或许不想为这么简单的任务安装任何第三方库。

6.3.1　实战演练

在这个简单的屏幕抓取示例中，我们使用谷歌地图URL查询一个城市的纬度和经度。在谷歌地图中搜索一次之后就能找到查询所需的URL。我们可以按照下面的步骤从谷歌地图中提取信息。

城市名使用argparse模块从命令行中获取。

地图搜索URL使用urllib模块中的urlopen()函数打开。如果URL正确，会得到一个XML格式的输出。

然后处理XML输出，获取该城市的地理坐标。

代码清单6-2使用谷歌地图查找一个城市的地理坐标，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 6
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import os
import urllib

ERROR_STRING = '<error>'

def find_lat_long(city):
    """ Find geographic coordinates """
    # Encode query string into Google maps URL
    url = 'http://maps.google.com/?q=' + urllib.quote(city) + '&output=js'
    print 'Query: %s' % (url)

    # Get XML location from Google maps
    xml = urllib.urlopen(url).read()

    if ERROR_STRING in xml:
        print '\nGoogle cannot interpret the city.'
        return
    else:
        # Strip lat/long coordinates from XML
        lat,lng = 0.0,0.0
        center = xml[xml.find('{center')+10:xml.find('}',xml.find('{center'))]
        center = center.replace('lat:','').replace('lng:','')
```

6

```
            lat,lng = center.split(',')
            print "Latitude/Longitude: %s/%s\n" %(lat, lng)

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='City Geocode Search')
    parser.add_argument('--city', action="store", dest="city", required=True)
    given_args = parser.parse_args()

    print "Finding geographic coordinates of %s" %given_args.city
    find_lat_long(given_args.city)
```

运行这个脚本后，会看到类似下面的输出：

```
$ python 6_2_geo_coding_by_google_maps.py --city=London
Finding geograhic coordinates of London
Query: http://maps.google.com/?q=London&output=js
Latitude/Longitude: 51.5112140000000002/-0.119824
```

6.3.2　原理分析

这个攻略从命令行中获取城市名，然后将其传给`find_lat_long()`函数。这个函数使用`urllib`模块中的`urlopen()`函数查询谷歌地图服务，得到XML格式的结果。然后，在得到的结果中搜索字符串`'<error>'`，如果找不到就说明结果没问题。

如果你把原始的XML打印出来，会发现有很多字符，这些字符是为浏览器生成的。在浏览器中，需要在地图上显示图层。但在这个攻略中，我们只需要纬度和经度。

我们使用字符串处理方法`find()`从原始的XML中提取出纬度和经度。我们搜索的关键字是“center”，以便找出地理坐标信息。但得到的结果中包含一些额外的字符，所以又调用`replace()`方法将其删除。

你可以使用这个攻略查找世界上任何一座城市的经纬度。

6.4　搜索维基百科中的文章

维基百科是个非常棒的网站，汇聚了几乎所有的信息，例如人物、场所和技术等。如果你想使用Python脚本在维基百科中搜索点儿什么，可以参考这个攻略。

下面是一篇文章示例：



6.4.1 准备工作

你需要使用pip或easy_install从PyPI上安装第三方库pyyaml，输入的命令为$ pip install pyyaml或$ easy_install pyyaml。

6.4.2 实战演练

我们使用关键字“Islam”搜索维基百科，然后把结果打印出来，一行显示一个。

代码清单6-3展示了如何在维基百科中搜索文章，如下所示：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Python Network Programming Cookbook -- Chapter - 6
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import re
import yaml
import urllib
import urllib2

SEARCH_URL =
'http://%s.wikipedia.org/w/api.php?action=query&list=search&srsearch=%s&sroffset=%d&srlimit=%d&format=yaml'

class Wikipedia:

    def __init__(self, lang='en'):
        self.lang = lang
```

```
    def _get_content(self, url):
        request = urllib2.Request(url)
        request.add_header('User-Agent', 'Mozilla/20.0')

        try:
            result = urllib2.urlopen(request)
        except urllib2.HTTPError, e:
            print "HTTP Error:%s" %(e.reason)
        except Exception, e:
            print "Error occured: %s" %str(e)
        return result

    def search_content(self, query, page=1, limit=10):
        offset = (page - 1) * limit
        url = SEARCH_URL % (self.lang, urllib.quote_plus(query), offset, limit)
        content = self._get_content(url).read()

        parsed = yaml.load(content)
        search = parsed['query']['search']
        if not search:
            return

        results = []
        for article in search:
            snippet = article['snippet']
            snippet = re.sub(r'(?m)<.*?>', '', snippet)
            snippet = re.sub(r'\s+', ' ', snippet)
            snippet = snippet.replace(' . ', '. ')
            snippet = snippet.replace(' , ', ', ')
            snippet = snippet.strip()

            results.append({
                'title' : article['title'].strip(),
                'snippet' : snippet
            })

        return results

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Wikipedia search')
    parser.add_argument('--query', action="store", dest="query", required=True)
    given_args = parser.parse_args()
    wikipedia = Wikipedia()
    search_term = given_args.query
    print "Searching Wikipedia for %s" %search_term
    results = wikipedia.search_content(search_term)
    print "Listing %s search results..." %len(results)
    for result in results:
        print "==%s== \n \t%s" %(result['title'], result['snippet'])
    print "---- End of search results ----"
```

运行这个脚本在维基百科中搜索“Islam”，得到的输出结果如下所示：

```
$ python 6_3_search_article_in_wikipedia.py --query='Islam'
```

```
Searching Wikipedia for Islam
Listing 10 search results...
==Islam==
     Islam. (•|ɪ | s | l |ɑ ː| m الإسلام , ar | ALA | al-ʾIslām æl? ɪ s•læːm | IPA |
ar-al_islam. ...
==Sunni Islam==
     Sunni Islam (• | s | uː | n | i or • | s | ʊ | n | i |) is the
largest branch of Islam ; its adherents are referred to in Arabic as ...
==Muslim==
     A Muslim, also spelled Moslem is an adherent of Islam, a
monotheistic Abrahamic religion based on the Qur'an —which Muslims
consider the ...
==Sharia==
     is the moral code and religious law of Islam. Sharia deals with
many topics addressed by secular law, including crime, politics, and ...
==History of Islam==
     The history of Islam concerns the Islamic religion and its
adherents, known as Muslim s. " "Muslim" is an Arabic word meaning
"one who ...
==Caliphate==
     a successor to Islamic prophet Muhammad ) and all the Prophets
of Islam. The term caliphate is often applied to successions of
Muslim ...
==Islamic fundamentalism==
     Islamic ideology and is a group of religious ideologies seen as
advocating a return to the "fundamentals" of Islam : the Quran and
the Sunnah. ...
==Islamic architecture==
     Islamic architecture encompasses a wide range of both secular
and religious styles from the foundation of Islam to the present day. ...
---- End of search results ----
```

6.4.3 原理分析

首先，我们组建了用于搜索文章的维基百科URL模板。然后定义一个名为`Wikipedia`的类，其中有两个方法：`_get_content()`和`search_content()`。默认情况下，初始化时，把语言属性`lang`设为`en`（英语）。

命令行中输入的查询字符串传给`search_content()`方法，替换模板中的语言、查询字符串、偏移页数和返回结果数量，得到真正的搜索URL。`search_content()`方法的`page`参数是可选的，偏移页数由表达式`(page -1) * limit`计算得出。

搜索结果的内容通过`_get_content()`方法获得。在`_get_content()`方法中调用了`urllib`模块中的`urlopen()`函数。在搜索URL中，我们把结果的格式设为`yaml`，这基本上就是纯文本文件。然后再使用Python的`pyyaml`库解析得到的`yaml`格式搜索结果。

搜索结果使用正则表达式替换各结果中的内容。例如，`re.sub(r'(?m)<.*?>', '', snippet)`在字符串`snippet`中替换匹配`(?m)<.*?>`模式的内容。若想进一步学习正则表达式，

请阅读Python文档，地址是http://docs.python.org/2/howto/regex.html。

在维基百科中，每篇文章都有一个摘要或简短说明。我们创建了一个由字典组成的列表，列表中的每个元素都包含一个搜索结果的标题和摘要。然后遍历这个由字典组成的列表，打印搜索结果。

6.5　使用谷歌搜索股价

如果你关注某公司的股价，这个攻略能帮助你了解该公司今天的股价。

6.5.1　准备工作

假设你知道所关注的公司在股票交易所挂牌上市使用的代号。如果不知道，可以在该公司的网站中查找，或者在谷歌中搜索。

6.5.2　实战演练

我们要使用谷歌财经（http://finance.google.com/）搜索指定公司的股价。你可以在命令行中输入代号，如下面的代码所示。

代码清单6-4说明如何在谷歌中搜索股价，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 6
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import urllib
import re
from datetime import datetime

SEARCH_URL = 'http://finance.google.com/finance?q='

def get_quote(symbol):
    content = urllib.urlopen(SEARCH_URL + symbol).read()
    m = re.search('id="ref_694653_l".*?>(.*?)<', content)
    if m:
        quote = m.group(1)
    else:
        quote = 'No quote available for: ' + symbol
    return quote

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Stock quote search')
```

```
    parser.add_argument('--symbol', action="store", dest="symbol", required=True)
    given_args = parser.parse_args()
    print "Searching stock quote for symbol '%s'" %given_args.symbol
    print "Stock  quote for %s at %s: %s" %(given_args.symbol , datetime.today(),
get_quote(given_args.symbol))
```

运行这个脚本后，会看到类似下面的输出。这里，我们输入代号goog，搜索谷歌的股价，如下所示：

```
$ python 6_4_google_stock_quote.py --symbol=goog
Searching stock quote for symbol 'goog'
Stock quote for goog at 2013-08-20 18:50:29.483380: 868.86
```

6.5.3 原理分析

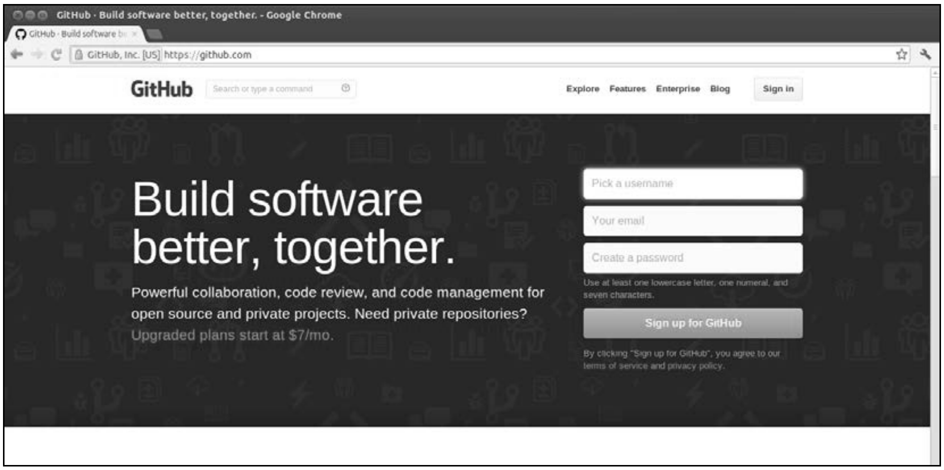
在这个脚本中，使用urllib模块中的urlopen()函数从谷歌财经网站中获取股票数据。

我们使用正则表达式库re，从第一组数据中获取股价。re库的search函数很强大，能搜索内容并过滤特定公司的ID。

我们使用这个攻略搜索了谷歌的股价，在2013年8月20日，其股价是868.86。

6.6 搜索 GitHub 中的源代码仓库

作为一个Python程序员，你可能已经知道GitHub（http://www.github.com，如下面的截图所示）这个源代码分享网站了。使用GitHub，你可以把源代码私下分享给团队，也可以公开分享给全世界。GitHub有一个好用的API接口，可以查询任何源代码仓库。这个攻略或许能为你的源代码搜索引擎提供一些起步代码。



6.6.1　准备工作

运行这个脚本需要安装Python第三方库requests，执行命令$ pip install requests或$ easy_install requests即可。

6.6.2　实战演练

我们要定义search_repository()函数，其参数是作者名（即程序员名）、仓库名和搜索的键名，得到的是键对应的结果。根据GitHub的API，可用的搜索键有：issues_url、has_wiki、forks_url、mirror_url、subscription_url、notifications_url、collaborators_url、updated_at、private、pulls_url、issue_comment_url、labels_url、full_name、owner、statuses_url、id、keys_url、description、tags_url、network_count、downloads_url、assignees_url、contents_url、git_refs_url、open_issues_count、clone_url、watchers_count、git_tags_url、milestones_url、languages_url、size、homepage、fork、commits_url、issue_events_url、archive_url、comments_url、events_url、contributors_url、html_url、forks、compare_url、open_issues、git_url、svn_url、merges_url、has_issues、ssh_url、blobs_url、master_branch、git_commits_url、hooks_url、has_downloads、watchers、name、language、url、created_at、pushed_at、forks_count、default_branch、teams_url、trees_url、organization、branches_url、subscribers_url和stargazers_url。

代码清单6-5给出了在GitHub中搜索源代码仓库详情的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 6
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

SEARCH_URL_BASE = 'https://api.github.com/repos'

import argparse
import requests
import json

def search_repository(author, repo, search_for='homepage'):
    url = "%s/%s/%s" %(SEARCH_URL_BASE, author, repo)
    print "Searching Repo URL: %s" %url
    result = requests.get(url)
    if(result.ok):
        repo_info = json.loads(result.text or result.content)
        print "Github repository info for: %s" %repo
        result = "No result found!"
        keys = []
        for key,value in repo_info.iteritems():
```

```
            if search_for in key:
                result = value
        return result

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Github search')
    parser.add_argument('--author', action="store", dest="author", required=True)
    parser.add_argument('--repo', action="store", dest="repo", required=True)
    parser.add_argument('--search_for', action="store", dest="search_for",
required=True)

    given_args = parser.parse_args()
    result = search_repository(given_args.author, given_args.repo,
given_args.search_for)
    if isinstance(result, dict):
        print "Got result for '%s'..." %(given_args.search_for)
        for key,value in result.iteritems():
            print "%s => %s" %(key,value)
    else:
        print "Got result for %s: %s" %(given_args.search_for, result)
```

如果运行这个脚本搜索Python Web框架Django的拥有者，得到的结果如下所示：

```
$ python 6_5_search_code_github.py --author=django --repo=django --search_for=owner
Searching Repo URL: https://api.github.com/repos/django/django
Github repository info for: django
Got result for 'owner'...
following_url => https://api.github.com/users/django/following{/other_user}
events_url => https://api.github.com/users/django/events{/privacy}
organizations_url => https://api.github.com/users/django/orgs
url => https://api.github.com/users/django
gists_url => https://api.github.com/users/django/gists{/gist_id}
html_url => https://github.com/django
subscriptions_url => https://api.github.com/users/django/subscriptions
avatar_url => https://1.gravatar.com/avatar/fd542381031aa84dca86628ece84f
c07?d=https%3A%2F%2Fidenticons.github.com%2Fe94df919e51ae96652259468415d4f77.png
repos_url => https://api.github.com/users/django/repos
received_events_url => https://api.github.com/users/django/received_events
gravatar_id => fd542381031aa84dca86628ece84fc07
starred_url => https://api.github.com/users/django/starred{/owner}{/repo}
login => django
type => Organization
id => 27804
followers_url => https://api.github.com/users/django/followers
```


6.6.3 原理分析

这个脚本接收三个命令行参数：仓库作者（`--author`）、仓库名（`--repo`）、要搜索的信息（`--search_for`）。这些参数由`argparse`模块解析。

在`search_repository()`函数中，把这些命令行参数添加到一个固定的搜索URL中，然后调用`requests`模块中的`get()`方法获取搜索结果。

默认情况下，返回的搜索结果是JSON格式。然后，调用json模块中的loads()方法处理搜索结果。在结果中查找搜索的键，把键对应的值返回给search_repository()函数的调用程序。

在__main__块中，我们检查搜索结果是否为一个Python字典实例。如果是，就遍历结果，把键值对打印出来；否则，直接打印结果。

6.7　读取 BBC 的新闻订阅源

如果你在开发一个新闻和故事相关的社会化网站，或许想显示世界上不同新闻通讯社（例如BBC和路透社）的新闻。让我们试着使用Python脚本从BBC读取新闻。

6.7.1　准备工作

这个攻略依赖于Python第三方库feedparser。你可以执行下面的命令安装这个库：

```
$ pip install feedparser
```

或

```
$ easy_install feedparser
```

6.7.2　实战演练

首先，我们要从BBC的网站上找到新闻订阅源的URL。这个URL可以作为搜索不同类型新闻的模板，例如国际新闻、国内新闻、健康新闻、商业新闻和技术新闻。新闻的类型可以从用户的输入中获取。然后，调用read_news()函数，从BBC的网站中读取新闻。

代码清单6-6说明了如何从BBC的新闻订阅源读取新闻，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 6
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

from datetime import datetime
import feedparser
BBC_FEED_URL = 'http://feeds.bbci.co.uk/news/%s/rss.xml'

def read_news(feed_url):
    try:
        data = feedparser.parse(feed_url)
    except Exception, e:
        print "Got error: %s" %str(e)

    for entry in data.entries:
```

```
        print(entry.title)
        print(entry.link)
        print(entry.description)
        print("\n")

if __name__ == '__main__':
    print "==== Reading technology news feed from bbc.co.uk (%s)===="
%datetime.today()
    print "Enter the type of news feed: "
    print "Available options are: world, uk, health, sci-tech, business, technology"
    type = raw_input("News feed type:")
    read_news(BBC_FEED_URL %type)
    print "==== End of BBC news feed ====="
```

运行这个脚本后，会显示可选的新闻类别。如果选择技术类，就会显示技术相关的最新新闻，如下面的输出所示：

```
$ python 6_6_read_bbc_news_feed.py
==== Reading technology news feed from bbc.co.uk (2013-08-20 19:02:33.940014)====
Enter the type of news feed:
Available options are: world, uk, health, sci-tech, business, technology
News feed type:technology
Xbox One courts indie developers
http://www.bbc.co.uk/news/technology-23765453#sa-ns_mchannel=rss&ns_source=PublicR
SS20-sa
Microsoft is to give away free Xbox One development kits to encourage
independent developers to self-publish games for its forthcoming console.

Fast in-flight wi-fi by early 2014
http://www.bbc.co.uk/news/technology-23768536#sa-ns_mchannel=rss&ns_source=PublicR
SS20-sa
Passengers on planes, trains and ships may soon be able to take advantage
of high-speed wi-fi connections, says Ofcom.

Anonymous 'hacks council website'
http://www.bbc.co.uk/news/uk-england-surrey-23772635#sa-ns_mchannel=rss&ns_source=
PublicRSS20-sa
A Surrey council blames hackers Anonymous after references to a Guardian
journalist's partner detained at Heathrow Airport appear on its website.

Amazon.com website goes offline
http://www.bbc.co.uk/news/technology-23762526#sa-ns_mchannel=rss&ns_source=PublicR
SS20-sa
Amazon's US website goes offline for about half an hour, the latest high-
profile internet firm to face such a problem in recent days.

[TRUNCATED]
```

6

6.7.3 原理分析

在这个攻略中，`read_news()`函数依赖于Python第三方模块`feedparser`。`feedparser`模

块中的parse()方法以结构化形式返回订阅源中的数据。

在这个攻略中，parse()方法解析指定的订阅源URL。这个URL由BBC_FEED_URL和用户的输入组成。

如果无异常就调用parse()方法获取订阅源中的数据，再把数据中的内容打印出来，例如每条新闻的标题、链接和描述。

6.8　爬取网页中的链接

有时你可能想在网页中查找某个关键字。在网页浏览器中，可以使用页内搜索功能找到关键字。有些浏览器还能高亮显示关键字。如果情况复杂，你或许还想进一步深入查找，跟踪网页中的每个URL，查找某个关键字。这个攻略的目的是自动化完成这样的任务。

6.8.1　实战演练

我们来定义search_links()函数，它接收三个参数：搜索的URL、递归搜索的深度、搜索关键字。因为每个URL对应的内容中都可能有很多链接，而且各链接指向的内容中或许还有更多的链接要爬取，所以我们要递归搜索。为了限制递归搜索，我们定义了一个深度。到达指定的深度后，就不会再继续搜索了。

代码清单6-7给出了爬取网页中链接的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 6
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import sys
import httplib
import re

processed = []

def search_links(url, depth, search):
    # Process http links that are not processed yet
    url_is_processed = (url in processed)
    if (url.startswith("http://") and (not url_is_processed)):
        processed.append(url)
        url = host = url.replace("http://", "", 1)
        path = "/"

        urlparts = url.split("/")
        if (len(urlparts) > 1):
```

```
            host = urlparts[0]
            path = url.replace(host, "", 1)

        # Start crawing
        print "Crawling URL path:%s%s " %(host, path)
        conn = httplib.HTTPConnection(host)
        req = conn.request("GET", path)
        result = conn.getresponse()

        # find the links
        contents = result.read()
        all_links = re.findall('href="(.*?)"', contents)

        if (search in contents):
            print "Found " + search + " at " + url

        print " ==> %s: processing %s links" %(str(depth), str(len(all_links)))
        for href in all_links:
            # Find relative urls
            if (href.startswith("/")):
                href = "http://" + host + href

            # Recurse links
            if (depth > 0):
                search_links(href, depth-1, search)
    else:
        print "Skipping link: %s ..." %url

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Webpage link crawler')
    parser.add_argument('--url', action="store", dest="url", required=True)
    parser.add_argument('--query', action="store", dest="query", required=True)
    parser.add_argument('--depth', action="store", dest="depth", default=2)

    given_args = parser.parse_args()

    try:
        search_links(given_args.url,  given_args.depth,given_args.query)
    except KeyboardInterrupt:
        print "Aborting search by user request."
```

如果运行这个脚本在www.python.org中搜索`python`，会看到类似下面的输出：

```
$ python 6_7_python_link_crawler.py --url='http://python.org' --query='python'
Crawling URL path:python.org/
Found python at python.org
 ==> 2: processing 123 links
Crawling URL path:www.python.org/channews.rdf
Found python at www.python.org/channews.rdf
 ==> 1: processing 30 links
Crawling URL path:www.python.org/download/releases/3.4.0/
Found python at www.python.org/download/releases/3.4.0/
 ==> 0: processing 111 links
Skipping link:
```

```
https://ep2013.europython.eu/blog/2013/05/15/epc20145-call-proposals ...
Crawling URL path:www.python.org/download/releases/3.2.5/
Found python at www.python.org/download/releases/3.2.5/
 ==> 0: processing 113 links
...
Skipping link: http://www.python.org/download/releases/3.2.4/ ...
Crawling URL path:wiki.python.org/moin/WikiAttack2013
^CAborting search by user request.
```

6.8.2　原理分析

这个攻略接收三个命令行参数：搜索的URL（`--url`）、查询字符串（`--query`）、递归深度（`--depth`）。这些参数由`argparse`模块解析。

把这三个参数传入`search_links()`函数后，会递归遍历在指定网页中找到的所有链接。如果运行很长时间还没结束，你应该提前退出脚本。因此，我们才把`search_links()`函数放在`try-except`块中，以便捕获用户在键盘中输入的中断操作，例如按Ctrl+C键。

`search_links()`函数把访问过的链接保存在`processed`列表中。`processed`放在全局作用域中，以便递归调用函数时使用。

每次搜索时，都要保证只处理HTTP URL，防止出现潜在的SSL验证错误。URL被分成主机和路径两部分。顶层爬取使用`httplib`库中的`HTTPConnection()`函数实例化。然后发起一个`GET`请求，使用正则表达式模块`re`处理响应，收集响应中的所有链接。然后在响应中查找要搜索的关键字。如果找到了关键字，就打印出来。

收集到的链接使用相同的方式递归访问。如果找到了相关链接，就在地址前加上`http://`，转换成完整的URL。如果搜索深度大于零，说明可以递归搜索，把深度减去一后，再次调用搜索函数。当搜索深度为零时，递归结束。

第 7 章

跨设备编程

本章攻略：

- 使用telnet在远程主机中执行shell命令
- 通过SFTP把文件复制到远程设备中
- 打印远程设备的CPU信息
- 在远程主机中安装Python包
- 在远程主机中运行MySQL命令
- 通过SSH把文件传输到远程设备中
- 远程配置Apache运行网站

7.1 简介

本章推荐一些有趣的Python库。这些攻略的目的是，向系统管理员和高级Python程序员介绍如何编写代码连接远程系统执行命令。本章首先介绍使用Python内置库`telnetlib`编写的简单攻略，然后介绍知名的远程连接库`paramiko`，最后介绍强大的远程系统管理库`fabric`。经常编写脚本完成自动化部署任务（例如部署Web应用或编译应用的二进制文件）的开发者很喜欢`fabric`库。

7.2 使用 telnet 在远程主机中执行 shell 命令

如果想通过telnet连接旧的网络交换机或路由器，无需使用bash脚本或交互式shell，可以在Python脚本中完成这一操作。这个攻略要创建一个简单的telnet会话，说明如何在远程主机中执行shell命令。

7.2.1 准备工作

你需要在自己的设备中安装telnet服务器，并确保其能正常运行。你可以使用操作系统专用的包管理器安装telnet服务器包。例如，在Debian/Ubuntu中，可以使用`apt-get`或`aptitude`安装

telnetd包，如下面的命令所示：

```
$ sudo apt-get install telnetd
$ telnet localhost
```

7.2.2　实战演练

我们来定义一个函数，从命令行中获取用户登录凭据，然后连接telnet服务器。

成功连接后，这个函数会把ls命令发送给服务器，然后显示命令的输出，例如列出一个目录中的内容。

代码清单7-1是一个telnet会话的代码，在远程主机中执行一个Unix命令，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 7
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import getpass
import sys
import telnetlib

HOST = "localhost"

def run_telnet_session():
    user = raw_input("Enter your remote account: ")
    password = getpass.getpass()

    session = telnetlib.Telnet(HOST)

    session.read_until("login: ")
    session.write(user + "\n")
    if password:
        session.read_until("Password: ")
        session.write(password + "\n")

    session.write("ls\n")
    session.write("exit\n")

    print session.read_all()

if __name__ == '__main__':
    run_telnet_session()
```

如果本地设备中运行有telnet服务器，运行这个脚本后，会要求你输入远程主机的用户账户和密码。在Debian设备中执行这个telnet会话得到的输出如下所示：

```
$ python 7_1_execute_remote_telnet_cmd.py
Enter remote hostname e.g. localhost: localhost
```

```
Enter your remote account: faruq
Password:

ls
exit
Last login: Mon Aug 12 10:37:10 BST 2013 from localhost on pts/9
Linux debian6 2.6.32-5-686 #1 SMP Mon Feb 25 01:04:36 UTC 2013 i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have new mail.
faruq@debian6:~$ ls
down               Pictures               Videos
Downloads          projects               yEd
Dropbox            Public
env                readme.txt
faruq@debian6:~$ exit
logout
```

7.2.3　原理分析

这个攻略使用Python内置的telnetlib网络库创建telnet会话。run_telnet_session()函数从命令行中获取用户名和密码。获取密码使用的是getpass模块中的getpass()函数，这个函数不会让你看到屏幕中输入的内容。

为了创建telnet会话，需要实例化Telnet类，初始化时要指定主机名参数。在这个攻略中，主机名是localhost。你可以使用argparse模块把主机名传给脚本。

telnet会话的远程输出可以使用read_until()方法获取。登录提示符就是使用这个方法检测到的。然后，使用write()方法把用户名和一个换行符发送给远程设备（在这个攻略中，把同一台设备当做远程主机）。再使用类似的方式把密码提供给远程主机。

然后，把ls密令发送给远程设备执行。最后，发送exit命令中断连接，再使用read_all()方法获取从远程主机中接收的全部会话数据，将其打印在屏幕上。

7

7.3　通过 SFTP 把文件复制到远程设备中

如果想安全地把本地设备中的文件上传或复制到远程设备中，可以使用“安全文件传输协议”（Secure File Transfer Protocol，简称SFTP）。

7.3.1　准备工作

这个攻略要使用一个强大的第三方网络库paramiko，演示如何使用SFTP复制文件，如下面的命令所示。paramiko的最新代码可以从GitHub（https://github.com/paramiko/paramiko）上获取，或者使用PyPI安装：

```
$ pip install paramiko
```

7.3.2　实战演练

这个攻略要从命令行中接收一些输入值，包括远程主机名、服务器端口、源文件名、目标文件名。简单起见，我们可以使用默认值或者硬编码的值。

连接远程服务器需要用户名和密码，这两个值可以从用户在命令行中的输入获取。

代码清单7-2说明了如何通过SFTP把文件复制到远程主机中，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 7
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import paramiko
import getpass

SOURCE = '7_2_copy_remote_file_over_sftp.py'
DESTINATION ='/tmp/7_2_copy_remote_file_over_sftp.py '

def copy_file(hostname, port, username, password, src, dst):
    client = paramiko.SSHClient()
    client.load_system_host_keys()
    print " Connecting to %s \n with username=%s... \n" %(hostname,username)
    t = paramiko.Transport((hostname, port))
    t.connect(username=username,password=password)
    sftp = paramiko.SFTPClient.from_transport(t)
    print "Copying file: %s to path: %s" %(src, dst)
    sftp.put(src, dst)
    sftp.close()
    t.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Remote file copy')
    parser.add_argument('--host', action="store", dest="host", default='localhost')
    parser.add_argument('--port', action="store", dest="port", default=22, type=int)
    parser.add_argument('--src', action="store", dest="src", default=SOURCE)
    parser.add_argument('--dst', action="store", dest="dst", default=DESTINATION)

    given_args = parser.parse_args()
```

```
    hostname, port =  given_args.host, given_args.port
    src, dst = given_args.src, given_args.dst

    username = raw_input("Enter the username:")
    password = getpass.getpass("Enter password for %s: " %username)

    copy_file(hostname, port, username, password, src, dst)
```

运行这个脚本后，会看到类似下面的输出：

```
$ python 7_2_copy_remote_file_over_sftp.py
Enter the username:faruq
Enter password for faruq:
 Connecting to localhost
 with username=faruq...
Copying file: 7_2_copy_remote_file_over_sftp.py to path:
/tmp/7_2_copy_remote_file_over_sftp.py
```

7.3.3 原理分析

这个攻略可以接收不同的输入值，然后连接到远程设备，通过SFTP复制文件。

这个攻略把命令行中的输入传给`copy_file()`函数，然后使用`paramiko`库中的`SSHClient`类创建一个SSH客户端。这个客户端需要加载系统的主机密钥。然后创建一个`Transport`类的实例，连接远程服务器。真正的SFTP连接对象`sftp`由`paramiko`库中的`SFTPClient.from_transport()`函数创建，其参数是`Transport`类的实例。

SFTP连接好之后，使用`put()`方法借由这个连接把本地文件复制到远程主机中。

最后，分别在各个对象上调用`close()`方法，清理SFTP连接和底层对象。这是一个好习惯。

7.4 打印远程设备的 CPU 信息

有时，我们需要通过SSH在远程设备中运行一个简单的命令。例如，查询远程设备的CPU或RAM信息。这种操作可以使用本节中的Python脚本完成。

7

7.4.1 准备工作

你需要安装第三方库`paramiko`，如下面的命令所示。`paramiko`的源代码可从GitHub仓库中获取，地址为https://github.com/paramiko/paramiko。

```
$ pip install paramiko
```

7.4.2　实战演练

我们可以使用paramiko模块创建一个远程会话连接Unix设备。

然后，通过这个会话，我们可以读取远程设备中的/proc/cpuinfo文件，获取CPU信息。

代码清单7-3是打印远程设备CPU信息的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 7
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import getpass
import paramiko

RECV_BYTES = 4096
COMMAND = 'cat /proc/cpuinfo'

def print_remote_cpu_info(hostname, port, username, password):
    client = paramiko.Transport((hostname, port))
    client.connect(username=username, password=password)

    stdout_data = []
    stderr_data = []
    session = client.open_channel(kind='session')
    session.exec_command(COMMAND)
    while True:
        if session.recv_ready():
            stdout_data.append(session.recv(RECV_BYTES))
        if session.recv_stderr_ready():
            stderr_data.append(session.recv_stderr(RECV_BYTES))
        if session.exit_status_ready():
            break

    print 'exit status: ', session.recv_exit_status()
    print ''.join(stdout_data)
    print ''.join(stderr_data)

    session.close()
    client.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Remote file copy')
    parser.add_argument('--host', action="store", dest="host", default='localhost')
    parser.add_argument('--port', action="store", dest="port", default=22, type=int)
    given_args = parser.parse_args()
    hostname, port =  given_args.host, given_args.port

    username = raw_input("Enter the username:")
    password = getpass.getpass("Enter password for %s: " %username)
    print_remote_cpu_info(hostname, port, username, password)
```

运行这个脚本后会显示指定主机的CPU信息，这里连接的是本地设备，如下所示：

```
$ python 7_3_print_remote_cpu_info.py
Enter the username:faruq
Enter password for faruq:
exit status:  0
processor: 0
vendor_id: GenuineIntel
cpu family: 6
model: 42
model name: Intel(R) Core(TM) i5-2400S CPU @ 2.50GHz
stepping: 7
cpu MHz: 2469.677
cache size: 6144 KB
fdiv_bug: no
hlt_bug: no
f00f_bug: no
coma_bug: no
fpu: yes
fpu_exception: yes
cpuid level: 5
wp: yes
flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush mmx fxsr sse sse2 syscall nx rdtscp lm constant_
tsc up pni monitor ssse3 lahf_lm
bogomips: 4939.35
clflush size: 64
cache_alignment: 64
address sizes: 36 bits physical, 48 bits virtual
power management:
```

7.4.3 原理分析

首先，收集连接所需的参数，例如主机名、端口号、用户名和密码。然后，把这些参数传给print_remote_cpu_info()函数。

在这个函数中，使用paramiko模块中的Transport类创建了一个SSH客户端会话。然后使用提供的用户名和密码连接远程设备。我们可以在SSH客户端上调用open_channel()方法创建一个原始通信会话。若想在远程主机中执行命令，可以使用exec_command()方法。

7

把命令发送给远程主机之后，可以通过封阻会话对象的recv_ready()事件来获取远程主机的响应。我们可以创建两个列表，stdout_data和stderr_data，用来存储远程主机的输出和错误消息。

命令在远程设备中退出时，可以使用exit_status_ready()方法检测到。接收到的远程会话数据使用join()方法串接起来。

最后，分别在各对象上调用close()方法，中断会话和客户端连接。

7.5　在远程主机中安装 Python 包

在前面几个攻略中，你可能注意到了，在远程主机中执行操作时，要写很多代码建立连接。为了提高执行效率，最好抽象这些代码，只向程序员开放相对高级的操作。在远程设备中执行命令时总是要建立连接，这个过程很繁琐，也浪费时间。

Fabric（http://fabfile.org/）这个Python第三方模块可以解决这个问题。它只开放了适当数量的API，能高效地和远程设备交互。

这个攻略举个简单的例子说明如何使用Fabric。

7.5.1　准备工作

首先，我们要安装Fabric。你可以使用Python包管理工具`pip`或`easy_install`安装，如下面的命令所示。Fabric依赖于`paramiko`模块，安装Fabric时会自动安装`paramiko`。

```
$ pip install fabric
```

这里，我们要通过SSH协议连接远程主机，所以必须在远程主机中运行SSH服务器。如果想在本机中测试（假装是连接到远程设备），可以在本地安装`openssh`服务器包。在Debian/Ubuntu中，可以使用包管理器`apt-get`安装，如下面的命令所示：

```
$ sudo apt-get install openssh-server
```

7.5.2　实战演练

下面这段代码说明了如何使用Fabric安装Python包。

代码清单7-4给出了在远程主机中安装Python包所需的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 7
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

from getpass import getpass
from fabric.api import settings, run, env, prompt

def remote_server():
    env.hosts = ['127.0.0.1']
    env.user = prompt('Enter user name: ')
    env.password = getpass('Enter password: ')

def install_package():
    run("pip install yolk")
```

Fabric脚本和普通Python脚本的运行方式不一样。使用fabric库定义的所有函数都要保存在一个名为fabfile.py的Python脚本中。在这个脚本中，没有传统的__main__指令。你可以使用Fabric API定义方法，然后使用命令行工具fab执行。因此，我们不使用python <script>.py运行Fabric脚本，而是在当前目录中创建一个fabfile.py脚本，然后执行fab one_function_name another_function_name。

那么，我们来按照下面命令中的方式创建fabfile.py脚本。为了简化操作，你可以创建文件快捷方式，或者把其他文件链接到fabfile.py脚本上。首先，删除前面创建的fabfile.py文件，然后创建一个指向fabfile的快捷方式：

```
$ rm -rf fabfile.py
$ ln -s 7_4_install_python_package_remotely.py fabfile.py
```

如果运行fabfile，在远程主机中安装Python包yolk后会生成如下输出：

```
$ ln -sfn 7_4_install_python_package_remotely.py fabfile.py
$ fab remote_server install_package
Enter user name: faruq
Enter password:
[127.0.0.1] Executing task 'install_package'
[127.0.0.1] run: pip install yolk
[127.0.0.1] out: Downloading/unpacking yolk
[127.0.0.1] out:   Downloading yolk-0.4.3.tar.gz (86kB):
[127.0.0.1] out:   Downloading yolk-0.4.3.tar.gz (86kB): 100%  86kB
[127.0.0.1] out:   Downloading yolk-0.4.3.tar.gz (86kB):
[127.0.0.1] out:   Downloading yolk-0.4.3.tar.gz (86kB): 86kB
downloaded
[127.0.0.1] out:   Running setup.py egg_info for package yolk
[127.0.0.1] out:     Installing yolk script to /home/faruq/env/bin
[127.0.0.1] out: Successfully installed yolk
[127.0.0.1] out: Cleaning up...
[127.0.0.1] out:

Done.
Disconnecting from 127.0.0.1... done.
```


7.5.3　原理分析

这个攻略演示了如何使用Python脚本在远程主机中执行系统管理任务。这个脚本中定义了两个函数。remote_server()函数设定Fabric的env环境变量，例如主机名、用户名和密码等。

另一个函数install_package()调用run()方法，其参数是在命令行中输入的命令。在这个脚本中，执行的命令是pip install yolk，即使用pip安装Python包yolk。和前面的攻略相比，使用Fabric在远程主机中运行命令更简单也更高效。

7.6　在远程主机中运行 MySQL 命令

如果你需要远程管理MySQL服务器，可以参考这个攻略。这个攻略会告诉你如何在Python脚本中向远程MySQL服务器发送数据库命令。如果要设置一个使用数据库后台的Web应用，这个攻略可以作为设置Web应用过程中的一部分。

7.6.1　准备工作

这个攻略也需要先安装Fabric。你可以使用Python包管理工具`pip`或`easy_install`安装，如下面的命令所示。Fabric依赖于`paramiko`模块，安装Fabric时会自动安装`paramiko`。

```
$ pip install fabric
```

这里，我们要通过SSH协议连接远程主机，所以必须在远程主机中运行SSH服务器。远程主机中还要运行MySQL服务器。在Debian/Ubuntu中，可以使用包管理器`apt-get`安装`openssh`和`mysql`服务器，如下面的命令所示：

```
$ sudo apt-get install openssh-server mysql-server
```

7.6.2　实战演练

我们要设定一些Fabric环境变量，再定义几个用于远程管理MySQL的函数。在这些函数中，我们不会直接使用可执行文件`mysql`，而是通过echo把SQL命令发送给`mysql`。这么做能确保正确地把参数传递给`mysql`可执行文件。

代码清单7-5给出了远程运行MySQL命令所需的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 7
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

from getpass import getpass
from fabric.api import run, env, prompt, cd

def remote_server():
    # Edit this list to include remote hosts
    env.hosts = ['127.0.0.1']
    env.user = prompt('Enter your system username: ')
    env.password = getpass('Enter your system user password: ')
    env.mysqlhost = 'localhost'
    env.mysqluser = prompt('Enter your db username: ')
    env.mysqlpassword = getpass('Enter your db user password: ')
    env.db_name = ''

def show_dbs():
```

```
    """ Wraps mysql show databases cmd"""
    q = "show databases"
    run("echo '%s' | mysql -u%s -p%s" %(q, env.mysqluser, env.mysqlpassword))

def run_sql(db_name, query):
    """ Generic function to run sql"""
    with cd('/tmp'):
        run("echo '%s' | mysql -u%s -p%s -D %s" %(query, env.mysqluser, env.mysqlpassword,
db_name))

def create_db():
    """Create a MySQL DB for App version"""
    if not env.db_name:
        db_name = prompt("Enter the DB name:")
    else:
        db_name = env.db_name
    run('echo "CREATE DATABASE %s default character set utf8 collate
utf8_unicode_ci;"|mysql --batch --user=%s --password=%s --host=%s'\
        % (db_name, env.mysqluser, env.mysqlpassword, env.mysqlhost), pty=True)

def ls_db():
    """ List a dbs with size in MB """
    if not env.db_name:
        db_name = prompt("Which DB to ls?")
    else:
        db_name = env.db_name
    query = """SELECT table_schema "DB Name",
        Round(Sum(data_length + index_length) / 1024 / 1024, 1) "DB Size in MB"
        FROM   information_schema.tables
        WHERE table_schema = \"%s\"
        GROUP  BY table_schema """ %db_name
    run_sql(db_name, query)

def empty_db():
    """ Empty all tables of a given DB """
    db_name = prompt("Enter DB name to empty:")
    cmd = """
    (echo 'SET foreign_key_checks = 0;';
    (mysqldump -u%s -p%s --add-drop-table --no-data %s |
     grep ^DROP);
     echo 'SET foreign_key_checks = 1;') | \
     mysql -u%s -p%s -b %s
    """ %(env.mysqluser, env.mysqlpassword, db_name, env.mysqluser, env.mysqlpassword,
db_name)
    run(cmd)
```

若想运行这个脚本，要创建一个快捷方式fabfile.py。在命令行中执行下面的命令可以完成这一操作：

```
$ ln -sfn 7_5_run_mysql_command_remotely.py fabfile.py
```

然后，可以使用fab可执行文件执行不同的操作。

7

下述命令显示了一个数据库列表（使用SQL查询show databases）：

```
$ fab remote_server show_dbs
```

下述命令会创建一个新的MySQL数据库。如果没有定义Fabric环境变量db_name，会显示一个提示符，要求输入目标数据库的名称。数据库使用以下SQL命令创建：CREATE DATABASE <database_name> default character set utf8 collate utf8_unicode_ci;。

```
$ fab remote_server create_db
```

这个Fabric命令显示数据库的大小：

```
$ fab remote_server ls_db()
```

下面这个Fabric命令使用可执行文件mysqldump和mysql清空数据库。这个函数的作用和数据库的TRUNCATE命令类似，只不过同时还会删除所有表。结果就像新建一个没有任何表的数据库一样。

```
$ fab remote_server empty_db()
```

各命令的输出如下：

```
$ fab remote_server show_dbs
[127.0.0.1] Executing task 'show_dbs'
[127.0.0.1] run: echo 'show databases' | mysql -uroot -p<DELETED>
[127.0.0.1] out: Database
[127.0.0.1] out: information_schema
[127.0.0.1] out: mysql
[127.0.0.1] out: phpmyadmin
[127.0.0.1] out:

Done.
Disconnecting from 127.0.0.1... done.


$ fab remote_server create_db
[127.0.0.1] Executing task 'create_db'
Enter the DB name: test123
[127.0.0.1] run: echo "CREATE DATABASE test123 default character set utf8
collate utf8_unicode_ci;"|mysql --batch --user=root --password=<DELETED>
--host=localhost

Done.
Disconnecting from 127.0.0.1... done.


$ fab remote_server show_dbs
[127.0.0.1] Executing task 'show_dbs'
[127.0.0.1] run: echo 'show databases' | mysql -uroot -p<DELETED>
[127.0.0.1] out: Database
[127.0.0.1] out: information_schema
```

```
[127.0.0.1] out: collabtive
[127.0.0.1] out: test123
[127.0.0.1] out: testdb
[127.0.0.1] out:

Done.
Disconnecting from 127.0.0.1... done.
```

7.6.3 原理分析

这个脚本定义了Fabric使用的几个函数。第一个函数remote_server()设定环境变量。本地回送IP（127.0.0.1）保存在主机列表中。本地系统的用户密码和MySQL登录密码通过getpass()方法获取。

另一个函数利用Fabric中的run()函数，把MySQL命令回显给mysql可执行文件，发送给远程MySQL服务器。

run_sql()函数是个通用函数，作为一个包装函数，可在其他函数中使用。例如，在empty_db()函数中调用了run_sql()函数执行SQL命令。这么做可以让代码变得更有序、更简洁。

7.7 通过 SSH 把文件传输到远程设备中

使用Fabric执行远程系统管理任务时，如果想通过SSH把本地设备中的文件传输到远程设备中，可以使用Fabric内置的get()和put()函数。这个攻略向你展示如何定义函数传输文件，并且在传输前后检查硬盘空间。

7.7.1 准备工作

这个攻略也需要先安装Fabric。你可以使用Python包管理工具pip或easy_install安装，如下面的命令所示：

```
$ pip install fabric
```

这里，我们要通过SSH协议连接远程主机，所以必须在远程主机中安装并运行SSH服务器。

7.7.2 实战演练

我们先要为Fabric设定环境变量，然后再定义两个函数，一个用于下载文件，另一个用于上传文件。

代码清单7-6是通过SSH把文件传输到远程设备中所需的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 7
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

from getpass import getpass
from fabric.api import local, run, env, get, put, prompt, open_shell

def remote_server():
    env.hosts = ['127.0.0.1']
    env.password = getpass('Enter your system password: ')
    env.home_folder = '/tmp'

def login():
    open_shell(command="cd %s" %env.home_folder)

def download_file():
    print "Checking local disk space..."
    local("df -h")
    remote_path = prompt("Enter the remote file path:")
    local_path = prompt("Enter the local file path:")
    get(remote_path=remote_path, local_path=local_path)
    local("ls %s" %local_path)

def upload_file():
    print "Checking remote disk space..."
    run("df -h")
    local_path = prompt("Enter the local file path:")
    remote_path = prompt("Enter the remote file path:")
    put(remote_path=remote_path, local_path=local_path)
    run("ls %s" %remote_path)
```

若想运行这个脚本，要创建一个快捷方式fabfile.py。在命令行中执行下面的命令可以完成这一操作：

```
$ ln -sfn 7_6_transfer_file_over_ssh.py fabfile.py
```

然后，可以使用fab可执行文件执行不同的操作。

首先，若想使用这个脚本登录远程服务器，可以运行下面这个Fabric函数：

```
$ fab remote_server login
```

执行上述命令后会看到一个类似shell的微型环境。然后，可以使用下面的命令把文件从远程服务器下载到本地设备中：

```
$ fab remote_server download_file
```

类似地，上传文件可以使用下面这个命令：

```
$ fab remote_server upload_file
```

在这个例子中，我们通过SSH连接本地设备。因此，你要在本地设备中安装SSH服务器才能运行这个脚本。不然，你可以修改remote_server()函数，改成连接到远程主机，如下所示：

```
$ fab remote_server login
[127.0.0.1] Executing task 'login'
Linux debian6 2.6.32-5-686 #1 SMP Mon Feb 25 01:04:36 UTC 2013 i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have new mail.
Last login: Wed Aug 21 15:08:45 2013 from localhost
cd /tmp
faruq@debian6:~$ cd /tmp
faruq@debian6:/tmp$

<CTRL+D>
faruq@debian6:/tmp$ logout

Done.
Disconnecting from 127.0.0.1... done.

$ fab remote_server download_file
[127.0.0.1] Executing task 'download_file'
Checking local disk space...
[localhost] local: df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1              62G   47G   12G  81% /
tmpfs                 506M     0  506M   0% /lib/init/rw
udev                  501M  160K  501M   1% /dev
tmpfs                 506M  408K  505M   1% /dev/shm
Z_DRIVE              1012G  944G   69G  94% /media/z
C_DRIVE               466G  248G  218G  54% /media/c
Enter the remote file path: /tmp/op.txt
Enter the local file path: .
[127.0.0.1] download: chapter7/op.txt <- /tmp/op.txt
[localhost] local: ls .
7_1_execute_remote_telnet_cmd.py   7_3_print_remote_cpu_info.py
7_5_run_mysql_command_remotely.py  7_7_configure_Apache_for_hosting_
website_remotely.py  fabfile.pyc  __init__.py  test.txt
7_2_copy_remote_file_over_sftp.py  7_4_install_python_package_
remotely.py  7_6_transfer_file_over_ssh.py      fabfile.py
index.html     op.txt       vhost.conf

Done.
Disconnecting from 127.0.0.1... done.
```

7.7.3　原理分析

在这个攻略中，用到了几个Fabric内置的函数，在本地设备和远程设备之间传输文件。local()函数在本地设备中执行操作，run()函数在远程设备中执行操作。

在上传和下载文件前最好先检查目标设备中的可用硬盘空间。

这个操作使用Unix命令df实现。源文件和目标文件的路径可以在命令行中指定，也可硬编码在源文件中，以备无人值守时自动执行。

7.8　远程配置 Apache 运行网站

Fabric函数可以以普通用户身份运行，也可以超级用户身份运行。如果想在远程Apache服务器上运行网站，需要管理员权限才能创建配置文件以及重启Web服务器。这个攻略介绍Fabric的sudo()函数，以超级用户身份在远程设备中执行命令。这里，我们要配置运行网站所需的Apache虚拟主机。

7.8.1　准备工作

这个攻略需要先在本地设备中安装Fabric。你可以使用Python包管理工具pip或easy_install安装，如下面的命令所示：

```
$ pip install fabric
```

这里，我们要通过SSH协议连接远程主机，所以必须在远程主机中安装并运行SSH服务器。远程主机中还需要安装和运行Apache Web服务器。在Debian/Ubuntu中，可以使用包管理器apt-get安装，如下面的命令所示：

```
$ sudo apt-get install openssh-server apache2
```

7.8.2　实战演练

首先，我们要知道Apache的安装路径和一些配置参数，例如Web服务器的用户、用户组、虚拟主机配置文件的路径和初始化脚本。这些参数可以定义为常量。

然后，定义两个函数，remote_server()和setup_vhost()，使用Fabric执行Apache配置任务。

代码清单7-7是远程配置Apache运行网站所需的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 7
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

from getpass import getpass
from fabric.api import env, put, sudo, prompt
from fabric.contrib.files import exists

WWW_DOC_ROOT = "/data/apache/test/"
WWW_USER = "www-data"
WWW_GROUP = "www-data"
APACHE_SITES_PATH = "/etc/apache2/sites-enabled/"
APACHE_INIT_SCRIPT = "/etc/init.d/apache2 "

def remote_server():
    env.hosts = ['127.0.0.1']
    env.user = prompt('Enter user name: ')
    env.password = getpass('Enter your system password: ')

def setup_vhost():
    """ Setup a test website """
    print "Preparing the Apache vhost setup..."

    print "Setting up the document root..."
    if exists(WWW_DOC_ROOT):
        sudo("rm -rf %s" %WWW_DOC_ROOT)
    sudo("mkdir -p %s" %WWW_DOC_ROOT)

    # setup file permissions
    sudo("chown -R %s.%s %s" %(env.user, env.user, WWW_DOC_ROOT))

    # upload a sample index.html file
    put(local_path="index.html", remote_path=WWW_DOC_ROOT)
    sudo("chown -R %s.%s %s" %(WWW_USER, WWW_GROUP, WWW_DOC_ROOT))

    print "Setting up the vhost..."
    sudo("chown -R %s.%s %s" %(env.user, env.user, APACHE_SITES_PATH))

    # upload a pre-configured vhost.conf
    put(local_path="vhost.conf", remote_path=APACHE_SITES_PATH)
    sudo("chown -R %s.%s %s" %('root', 'root', APACHE_SITES_PATH))

    # restart Apache to take effect
    sudo("%s restart" %APACHE_INIT_SCRIPT)
    print "Setup complete. Now open the server path http://abc.remote-server.org/ in
your web browser."
```

为了运行这个脚本，要把下面这行加入主机文件中，例如/etc/hosts：

```
127.0.0.1 abc.remote-server.org abc
```

还要创建快捷方式fabfile.py。在命令行中，可以使用下面的命令完成：

```
$ ln -sfn 7_7_configure_Apache_for_hosting_website_remotely.py fabfile.py
```

然后，可以使用fab可执行文件执行不同的操作。

首先，若想使用这个脚本登录远程服务器，可以运行下面这个Fabric函数。得到的输出结果如下：

```
$ fab remote_server setup_vhost
[127.0.0.1] Executing task 'setup_vhost'
Preparing the Apache vhost setup...
Setting up the document root...
[127.0.0.1] sudo: rm -rf /data/apache/test/
[127.0.0.1] sudo: mkdir -p /data/apache/test/
[127.0.0.1] sudo: chown -R faruq.faruq /data/apache/test/
[127.0.0.1] put: index.html -> /data/apache/test/index.html
[127.0.0.1] sudo: chown -R www-data.www-data /data/apache/test/
Setting up the vhost...
[127.0.0.1] sudo: chown -R faruq.faruq /etc/apache2/sites-enabled/
[127.0.0.1] put: vhost.conf -> /etc/apache2/sites-enabled/vhost.conf
[127.0.0.1] sudo: chown -R root.root /etc/apache2/sites-enabled/
[127.0.0.1] sudo: /etc/init.d/apache2 restart
[127.0.0.1] out: Restarting web server: apache2apache2: Could not
reliably determine the server's fully qualified domain name, using
127.0.0.1 for ServerName
[127.0.0.1] out:  ... waiting apache2: Could not reliably determine the
server's fully qualified domain name, using 127.0.0.1 for ServerName
[127.0.0.1] out: .
[127.0.0.1] out:

Setup complete. Now open the server path http://abc.remote-server.org/ in your web
browser.

Done.
Disconnecting from 127.0.0.1... done.
```

运行这个脚本之后，你可以打开浏览器，访问主机文件（例如/etc/hosts）中设定的路径。在浏览器中会看到如下输出：

```
It works!
This is the default web page for this server.
The web server software is running but no content has been added, yet.
```

7.8.3　原理分析

这个攻略把初始的Apache配置参数设置为常量,然后定义了两个函数。在remote_server()函数中，和往常一样，设定了Fabric环境参数，例如主机、用户名和密码等。

setup_vhost()函数执行了一系列需要特殊权限的命令。首先，使用exists()函数检查是否已经创建了网站的文档根目录。如果该路径存在，就将其删除，在下一步中重新创建。然后再

执行chown命令，确保当前用户有权访问这个路径。

接着，把一个骨架HTML文件（index.html）上传到文档根路径中。上传后，再把文件的访问权限赋予Web服务器用户。

设置好文档根目录后，setup_vhost()函数把vhost.conf文件上传到Apache的网站配置路径中，然后把这个文件的拥有者设为根用户。

最后，这个脚本重启Apache服务器，让配置生效。如果配置正确，你会在浏览器中看到前面访问http://abc.remote-server.org/时显示的内容。

第 8 章

使用Web服务：XML-RPC、SOAP和REST

本章攻略：

- 查询本地XML-RPC服务器
- 编写一个多线程、多调用XML-RPC服务器
- 运行一个支持HTTP基本认证的XML-RPC服务器
- 使用REST从Flickr中收集一些照片信息
- 找出亚马逊S3 Web服务支持的SOAP方法
- 使用谷歌搜索定制信息
- 通过商品搜索API在亚马逊中搜索图书

8.1 简介

本章介绍一些有趣的Python攻略，通过三种不同的方式使用Web服务。这三种方式是“XML远程过程调用”（XML Remote Procedure Call，简称XML-RPC）、“简单对象访问协议”（Simple Object Access Protocol，简称SOAP）和“表现层状态转化”（Representational State Transfer，简称REST）。Web服务的目的，是让两个软件组件通过精心设计的协议在网络中交互。接口是机器可读的。多种不同的协议为Web服务的使用提供了便利。

本章包含这三种常用协议的示例。XML-RPC使用HTTP作为传输媒介，使用XML格式的内容通信。实现XML-RPC的服务器等待适配的客户端调用。客户端使用不同的参数调用服务器，执行远程过程。XML-RPC较为简单，但安全性不高。SOAP有一组丰富的协议，用于增强远程过程调用。REST是一种架构风格，让Web服务变得简单。REST架构中的操作使用HTTP请求方法完成，即`GET`、`POST`、`PUT`和`DELETE`。本章介绍这些Web服务协议和风格的用法，完成一些常见任务。

8.2　查询本地 XML-RPC 服务器

如果你经常做Web编程，可能会遇到这样的任务：从支持XML-RPC服务的网站中获取一些信息。在深入介绍XML-RPC服务之前，我们先架设一个XML-RPC服务器并和它通信。

8.2.1　准备工作

这个攻略要使用Python Supervisor程序。Supervisor广泛用于启动和管理可执行程序。Supervisor可以作为后台守护进程运行，能监控子进程，且子进程意外退出后能重启子进程。执行下面的命令即可安装Supervisor：

```
$ pip install supervisor
```

8.2.2　实战演练

我们要为Supervisor创建一个配置文件。这个攻略中提供了一个示例配置，定义了一个Unix HTTP服务器套接字和一些其他参数。注意`rpcinterface:supervisor`部分，其中`rpcinterface_factory`是用来和客户端通信的。

在`program:8_2_ multithreaded_multicall_xmlrpc_server.py`部分，我们使用Supervisor配置了一个简单的服务器程序，指定了命令和一些其他参数。

代码清单8-1a是一个简单的Supervisor配置，如下所示：

```
[unix_http_server]
file=/tmp/supervisor.sock   ; (the path to the socket file)
chmod=0700                  ; socket file mode (default 0700)

[supervisord]
logfile=/tmp/supervisord.log
loglevel=info
pidfile=/tmp/supervisord.pid
nodaemon=true

[rpcinterface:supervisor]
supervisor.rpcinterface_factory = supervisor.rpcinterface:make_main_rpcinterface

[program:8_2_multithreaded_multicall_xmlrpc_server.py]
command=python 8_2_multithreaded_multicall_xmlrpc_server.py ; the program (relative
uses PATH, can take args)
process_name=%(program_name)s ; process_name expr  (default %(program_name)s)
```

如果在你最喜欢的编辑器中创建上述Supervisor配置文件，调用这个文件就可以运行Supervisor。

现在，我们可以编写一个XML-RPC客户端，作为Supervisor的代理，获取运行中的进程信息。

8

代码清单8-1b是查询XML-RPC本地服务器的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 8
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import supervisor.xmlrpc
import xmlrpclib

def query_supervisr(sock):
    transport = supervisor.xmlrpc.SupervisorTransport(None, None,
                'unix://%s' %sock)
    proxy = xmlrpclib.ServerProxy('http://127.0.0.1',
            transport=transport)
    print "Getting info about all running processes via Supervisord..."
    print proxy.supervisor.getAllProcessInfo()

if __name__ == '__main__':
    query_supervisr(sock='/tmp/supervisor.sock')
```

运行这个Supervisor守护进程，会看到类似下面的输出：

```
chapter8$ supervisord
2013-09-27 16:40:56,861 INFO RPC interface 'supervisor' initialized
2013-09-27 16:40:56,861 CRIT Server 'unix_http_server' running
without any HTTP authentication checking
2013-09-27 16:40:56,861 INFO supervisord started with pid 27436
2013-09-27 16:40:57,864 INFO spawned:
'8_2_multithreaded_multicall_xmlrpc_server.py' with pid 27439
2013-09-27 16:40:58,940 INFO success:
8_2_multithreaded_multicall_xmlrpc_server.py entered RUNNING state,
process has stayed up for > than 1 seconds (startsecs)
```

注意，运行后启动了子进程`8_2_multithreaded_multicall_xmlrpc_server.py`。

现在，如果运行客户端代码，它会查询Supervisor的XML-RPC服务器接口，列出运行中的进程，如下所示：

```
$ python 8_1_query_xmlrpc_server.py
Getting info about all running processes via Supervisord...
[{'now': 1380296807, 'group':
'8_2_multithreaded_multicall_xmlrpc_server.py', 'description': 'pid
27439, uptime 0:05:50', 'pid': 27439, 'stderr_logfile':
'/tmp/8_2_multithreaded_multicall_xmlrpc_server.py-stderr---
supervisor-i_VmKz.log', 'stop': 0, 'statename': 'RUNNING', 'start':
1380296457, 'state': 20, 'stdout_logfile':
'/tmp/8_2_multithreaded_multicall_xmlrpc_server.py-stdout---
supervisor-eMuJqk.log', 'logfile':
'/tmp/8_2_multithreaded_multicall_xmlrpc_server.py-stdout---
supervisor-eMuJqk.log', 'exitstatus': 0, 'spawnerr': '', 'name':
'8_2_multithreaded_multicall_xmlrpc_server.py'}]
```

8.2.3 原理分析

这个攻略要在后台运行Supervisor守护进程（通过`rpcinterface`配置）。Supervisor会启动另一个XML-RPC服务器，即`8_2_ multithreaded_multicall_xmlrpc_server.py`。

客户端代码中定义了`query_supervisr()`方法，其参数是Supervisor套接字。在这个方法中，使用Unix套接字路径创建了一个`SupervisorTransport`实例。然后，实例化`xmlrpclib`模块中的`ServerProxy`类创建了一个XML-RPC服务器代理。传给`ServerProxy`类构造方法的参数是服务器地址和前面创建的`transport`。

随后，XML-RPC服务器代理调用Supervisor中的`getAllProcessInfo()`方法，把子进程的信息打印出来。打印的信息包括`pid`、`statename`和`description`等。

8.3 编写一个多线程、多调用 XML-RPC 服务器

你可以让你的XML-RPC服务器同时接受多个调用。这意味着，多个函数调用可以只返回一个结果。而且，如果服务器支持多线程，服务器启动后还能在单个线程中执行更多的代码。此时，程序的主线程处于非阻塞模式中。

8.3.1 实战演练

我们可以定义一个`ServerThread`类，继承自`threading.Thread`类，而且还可以把这个类的一个属性设为`SimpleXMLRPCServer`实例。这样就能接受多个调用了。

然后，我们可以定义两个函数：一个用来启动多线程、多调用XML-RPC服务器，另一个用于创建连接服务器的客户端。

代码清单8-2是编写多线程、多调用XML-RPC服务器所需的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 8
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import xmlrpclib
import threading

from SimpleXMLRPCServer import SimpleXMLRPCServer

# some trivial functions
def add(x,y):
    return x+y
```

8

```
def subtract(x, y):
    return x-y

def multiply(x, y):
    return x*y

def divide(x, y):
    return x/y

class ServerThread(threading.Thread):
    def __init__(self, server_addr):
        threading.Thread.__init__(self)
        self.server = SimpleXMLRPCServer(server_addr)
        self.server.register_multicall_functions()
        self.server.register_function(add, 'add')
        self.server.register_function(subtract, 'subtract')
        self.server.register_function(multiply, 'multiply')
        self.server.register_function(divide, 'divide')

    def run(self):
        self.server.serve_forever()

def run_server(host, port):
    # server code
    server_addr = (host, port)
    server = ServerThread(server_addr)
    server.start() # The server is now running
    print "Server thread started. Testing the server..."

def run_client(host, port):
    # client code
    proxy = xmlrpclib.ServerProxy("http://%s:%s/" %(host, port))
    multicall = xmlrpclib.MultiCall(proxy)
    multicall.add(7,3)
    multicall.subtract(7,3)
    multicall.multiply(7,3)
    multicall.divide(7,3)
    result = multicall()
    print "7+3=%d, 7-3=%d, 7*3=%d, 7/3=%d" % tuple(result)

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Multithreaded multicall XMLRPC
Server/Proxy')
    parser.add_argument('--host', action="store", dest="host", default='localhost')
    parser.add_argument('--port', action="store", dest="port", default=8000,
type=int)
    # parse arguments
    given_args = parser.parse_args()
    host, port =  given_args.host, given_args.port
    run_server(host, port)
    run_client(host, port)
```

运行这个脚本后，会看到类似下面的输出：

```
$ python 8_2_multithreaded_multicall_xmlrpc_server.py --port=8000
Server thread started. Testing the server...
localhost - - [25/Sep/2013 17:38:32] "POST / HTTP/1.1" 200 -
7+3=10, 7-3=4, 7*3=21, 7/3=2
```

8.3.2　原理分析

在这个攻略中，我们定义了ServerThread类，继承自Python线程库中的Thread类。这个子类初始化时把server属性设为一个SimpleXMLRPCServer服务器实例。XML-RPC服务器的地址可以在命令行中指定。为了启用多调用功能，我们在服务器实例上调用了register_multicall_functions()方法。

然后把四个简单的函数注册到XML-RPC服务器上：add()、subtract()、multiply()和divide()。这几个函数的作用正如其名称所示。

为了启动服务器，要把主机和端口传给run_server()函数。在这个函数中，使用前面说明的ServerThread类创建了一个服务器实例。然后在这个服务器实例上调用start()方法启动XML-RPC服务器。

再看客户端。run_client()函数同样从命令行中获取主机和端口。然后使用xmlrpclib中的ServerProxy类创建一个XML-RPC服务器代理实例。再把这个代理实例传给MultiCall类，创建一个实例multicall。现在，可以运行前面定义的四个RPC方法了，即add()、subtract()、multiply()和divide()。最后，我们只通过一次调用（multicall()）获取结果，再把结果中的元组在一行中打印出来。

8.4　运行一个支持 HTTP 基本认证的 XML-RPC 服务器

有时，你需要在XML-RPC服务器中实现认证功能。这个攻略介绍了一个简单的示例，说明如何在XML-RPC服务器中实现HTTP基本认证。

8.4.1　实战演练

我们可以定义一个SimpleXMLRPCServer类的子类，重新定义请求处理方法，以便当请求进入时，和指定的登录凭据比对。

8

代码清单8-3a是在XML-RPC服务器中实现HTTP基本认证的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 8
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.
```

```
import argparse
import xmlrpclib
from base64 import b64decode
from SimpleXMLRPCServer  import SimpleXMLRPCServer, SimpleXMLRPCRequestHandler

class SecureXMLRPCServer(SimpleXMLRPCServer):

    def __init__(self, host, port, username, password, *args, **kargs):
        self.username = username
        self.password = password
        # authenticate method is called from inner class
        class VerifyingRequestHandler(SimpleXMLRPCRequestHandler):
            # method to override
            def parse_request(request):
                if\SimpleXMLRPCRequestHandler.parse_request(request):
                    # authenticate
                    if self.authenticate(request.headers):
                        return True
                    else:
                        # if authentication fails return 401
                        request.send_error(401, 'Authentication failed, Try agin.')
                return False
        # initialize
        SimpleXMLRPCServer.__init__(self, (host, port),
requestHandler=VerifyingRequestHandler, *args, **kargs)

    def authenticate(self, headers):
        headers = headers.get('Authorization').split()
        basic, encoded = headers[0], headers[1]
        if basic != 'Basic':
            print 'Only basic authentication supported'
            return False
        secret = b64decode(encoded).split(':')
        username, password = secret[0], secret[1]
        return True if (username == self.username and password == self.password) else
False

def run_server(host, port, username, password):
    server = SecureXMLRPCServer(host, port, username, password)
    # simple test function
    def echo(msg):
        """Reply client in  uppser case """
        reply = msg.upper()
        print "Client said: %s. So we echo that in uppercase: %s" %(msg, reply)
        return reply
    server.register_function(echo, 'echo')
    print "Running a HTTP auth enabled XMLRPC server on %s:%s..." %(host, port)
    server.serve_forever()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Multithreaded multicall XMLRPC
Server/Proxy')
    parser.add_argument('--host', action="store", dest="host", default='localhost')
```

```
    parser.add_argument('--port', action="store", dest="port", default=8000, type=int)
    parser.add_argument('--username', action="store", dest="username", default='user')
    parser.add_argument('--password', action="store", dest="password", default='pass')
    # parse arguments
    given_args = parser.parse_args()
    host, port =  given_args.host, given_args.port
    username, password = given_args.username, given_args.password
    run_server(host, port, username, password)
```

运行这个脚本后，默认会看到如下输出：

```
$ python 8_3a_xmlrpc_server_with_http_auth.py
Running a HTTP auth enabled XMLRPC server on localhost:8000...
Client said: hello server.... So we echo that in uppercase: HELLO
SERVER...
localhost - - [27/Sep/2013 12:08:57] "POST /RPC2 HTTP/1.1" 200 -
```

现在，我们来创建一个简单的客户端代理，使用的登录凭据和服务器一样。

代码清单8-3b是XML-RPC客户端的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 8
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import xmlrpclib

def run_client(host, port, username, password):
    server = xmlrpclib.ServerProxy('http://%s:%s@%s:%s' %(username, password, host,
port, ))
    msg = "hello server..."
    print "Sending message to server: %s  " %msg
    print "Got reply: %s" %server.echo(msg)

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Multithreaded multicall XMLRPC
Server/Proxy')
    parser.add_argument('--host', action="store", dest="host", default='localhost')
    parser.add_argument('--port', action="store", dest="port", default=8000,
type=int)
    parser.add_argument('--username', action="store", dest="username",
default='user')
    parser.add_argument('--password', action="store", dest="password",
default='pass')
    # parse arguments
    given_args = parser.parse_args()
    host, port =  given_args.host, given_args.port
    username, password = given_args.username, given_args.password
    run_client(host, port, username, password)
```

8

运行这个脚本后，会看到如下输出：

```
$ python 8_3b_xmprpc_client.py
Sending message to server: hello server...
Got reply: HELLO SERVER...
```

8.4.2　原理分析

在服务器脚本中，继承SimpleXMLRPCServer类定义了SecureXMLRPCServer子类。在这个子类的初始化代码中，定义了用于拦截请求的VerifyingRequestHandler类，使用authenticate()方法做基本认证。

我们把HTTP请求的首部传给authenticate()方法，检查Authorization首部的值。如果值等于Basic，就使用base64标准模块中的b64decode()方法解码编码后的密码。提取出用户名和密码后，再和服务器指定的登录凭据比对。

在run_server()函数中定义了一个简单的echo()子函数，将其注册到SecureXMLRPCServer实例上。

在客户端脚本中，run_client()函数的参数是服务器地址和登录凭据，再把这些参数传给ServerProxy类构造方法，创建一个实例。然后使用echo()方法发送一行消息。

8.5　使用 REST 从 Flickr 中收集一些照片信息

很多网站都通过REST API提供Web服务接口。Flickr这个著名的照片分享网站就提供了REST接口。让我们来收集一些照片信息，构建一个特殊的数据库或者开发照片相关的应用。

8.5.1　实战演练

我们需要REST URL发起HTTP请求。简单起见，这些URL硬编码在脚本中。我们可以使用第三方模块requests发起REST请求。requests模块提供了一些便利的方法，包括get()、post()、put()和delete()。

为了和Flickr的Web服务通信，你需要注册一个账户，然后获取API密钥。API密钥可写入local_settings.py文件，或者在命令行中提供。

代码清单8-4是使用REST从Flickr中收集照片信息所需的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 8
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.
# Supply Flickr API key via local_settings.py

import argparse
import json
```

```
import requests

try:
    from local_settings import flickr_apikey
except ImportError:
    pass

def collect_photo_info(api_key, tag, max_count):
    """Collects some interesting info about some photos from Flickr.com for a given
tag """
    photo_collection = []
    url =  "http://api.flickr.com/services/rest/?method=flickr.photos.search&tags=%
s&format=json&nojsoncallback=1&api_key=%s" %(tag, api_key)
    resp = requests.get(url)
    results = resp.json()
    count  = 0
    for p in results['photos']['photo']:
        if count >= max_count:
            return photo_collection
        print 'Processing photo: "%s"' % p['title']
        photo = {}
        url = "http://api.flickr.com/services/rest/?method=flickr.photos.getInfo&
photo_id= " + p['id'] + "&format=json&nojsoncallback=1&api_key=" + api_key
        info = requests.get(url).json()
        photo["flickrid"] = p['id']
        photo["title"] = info['photo']['title']['_content']
        photo["description"] = info['photo']['description']['_content']
        photo["page_url"] = info['photo']['urls']['url'][0]['_content']

        photo["farm"] = info['photo']['farm']
        photo["server"] = info['photo']['server']
        photo["secret"] = info['photo']['secret']

        # comments
        numcomments = int(info['photo']['comments']['_content'])
        if numcomments:
            #print "   Now reading comments (%d)..." % numcomments
            url = "http://api.flickr.com/services/rest/?method=flickr.photos.comments.
getList&photo_id=" + p['id'] + "&format=json&nojsoncallback=1&api_key=" + api_key
            comments = requests.get(url).json()
            photo["comment"] = []
            for c in comments['comments']['comment']:
                comment = {}
                comment["body"] = c['_content']
                comment["authorid"] = c['author']
                comment["authorname"] = c['authorname']
                photo["comment"].append(comment)
        photo_collection.append(photo)
        count = count + 1
    return photo_collection

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Get photo info from Flickr')
    parser.add_argument('--api-key', action="store", dest="api_key",
```

```
default=flickr_apikey)
    parser.add_argument('--tag', action="store", dest="tag", default='Python')
    parser.add_argument('--max-count', action="store", dest="max_count", default=3,
type=int)
    # parse arguments
    given_args = parser.parse_args()
    api_key, tag, max_count =  given_args.api_key, given_args.tag,
given_args.max_count
    photo_info = collect_photo_info(api_key, tag, max_count)
    for photo in photo_info:
        for k,v in photo.iteritems():
            if k == "title":
                print "Showiing photo info...."
            elif k == "comment":
                "\tPhoto got %s comments." %len(v)
            else:
                print "\t%s => %s" %(k,v)
```

Flickr API密钥可以放在local_settings.py文件中，也可在命令行中提供（通过`--api-key`参数）。除了API密钥，还可指定搜索标签和结果的最大数量。默认情况下，这个脚本搜索`Python`标签，结果限制为3个，如下面的输出所示：

```
$ python 8_4_get_flickr_photo_info.py
Processing photo: "legolas"
Processing photo: ""The Dance of the Hunger of Kaa""
Processing photo: "Rocky"
    description => Stimson Python
Showiing photo info....
    farm => 8
    server => 7402
    secret => 6cbae671b5
    flickrid => 10054626824
    page_url => http://www.flickr.com/photos/102763809@N03/10054626824/
    description => &quot; 'Good. Begins now the dance--the Dance of the
Hunger of Kaa. Sit still and watch.'

He turned twice or thrice in a big circle, weaving his head from right to left.
Then he began making loops and figures of eight with his body, and soft,
oozy triangles that melted into squares and five-sided figures, and
coiled mounds, never resting, never hurrying, and never stopping his
low humming song. It grew darker and darker, till at last the dragging,
shifting coils disappeared, but they could hear the rustle of the
scales.&quot;
(From &quot;Kaa's Hunting&quot; in &quot;The Jungle Book&quot; (1893) by Rudyard
Kipling)

These old abandoned temples built around the 12th century belong to the
abandoned city which inspired Kipling's Jungle Book.
They are rising at the top of a mountain which dominates the jungle at
811 meters above sea level in the centre of the jungle of Bandhavgarh
located in the Indian state Madhya Pradesh.
Baghel King Vikramaditya Singh abandoned Bandhavgarh fort in 1617 when
Rewa, at a distance of 130 km was established as a capital.
```

```
Abandonment allowed wildlife development in this region.
When Baghel Kings became aware of it, he declared Bandhavgarh as their
hunting preserve and strictly prohibited tree cutting and wildlife
hunting...

Join the photographer at <a href="http://www.facebook.com/laurent.
goldstein.photography" rel="nofollow">www.facebook.com/laurent.goldstein.
photography</a>

© All photographs are copyrighted and all rights reserved.
Please do not use any photographs without permission (even for private
use).
The use of any work without consent of the artist is PROHIBITED and will
lead automatically to consequences.
Showiing photo info....
    farm => 6
    server => 5462
    secret => 6f9c0e7f83
    flickrid => 10051136944
    page_url => http://www.flickr.com/photos/designldg/10051136944/
    description => Ball Python
Showiing photo info....
    farm => 4
    server => 3744
    secret => 529840767f
    flickrid => 10046353675
    page_url => http://www.flickr.com/photos/megzzdollphotos/10046353675/
```

8.5.2 原理分析

这个攻略演示了如何使用Flickr的REST API和它交互。在这个示例中，`collect_photo_info()`函数有三个参数：Flickr API密钥、搜索标签和想要得到的搜索结果数量。

我们构建的第一个URL用于搜索照片。注意，在这个URL中，`method`参数的值是`flickr.photos.search`，希望得到的结果格式是JSON。

第一次调用`get()`方法得到的结果存储在变量`resp`中，然后在这个变量上调用`json()`方法，将其转换成JSON格式。然后在一个循环中读取`['photos']['photo']`中的JSON数据。得到的信息保存在`photo_collection`列表中。在这个列表中，每张照片的信息由一个字典表示。字典的键从前面的JSON格式响应中提取，照片的信息从另一个`GET`请求中获取。

注意，若想获取照片的评论，要再发起一个`GET`请求，从JSON格式响应中的`['comments']['comment']`元素获取。最后，把这些评论插入一个列表中，再添加到照片对应的字典里。

在`__main__`块中，我们从`photo_collection`列表中读取各字典，打印各张照片的一些有用信息。

8.6　找出亚马逊 S3 Web 服务支持的 SOAP 方法

如果你需要与实现了简单对象访问协议（SOAP）的Web服务交互，可以参考这个攻略。

8.6.1　准备工作

在这个攻略中我们可以使用第三方库SOAPpy，执行下面的命令即可安装这个库：

```
$ pip install SOAPpy
```

8.6.2　实战演练

我们要创建一个代理对象，在调用服务器上的方法之前，先找出支持哪些方法。

在这个攻略中，我们要和亚马逊S3存储服务交互。我们已经有了Web服务API的测试URL。你需要有API密钥才能执行这个简单的任务。

代码清单8-5是搜索亚马逊S3 Web服务支持的SOAP方法所需的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 8
# This program requires Python 2.7 or any later version

import SOAPpy

TEST_URL = 'http://s3.amazonaws.com/ec2-downloads/2009-04-04.ec2.wsdl'

def list_soap_methods(url):
    proxy = SOAPpy.WSDL.Proxy(url)
    print '%d methods in WSDL:' % len(proxy.methods) + '\n'
    for key in proxy.methods.keys():
        print "Key Name: %s" %key
        print "Key Details:"
        for k,v in proxy.methods[key].__dict__.iteritems():
            print "%s ==> %s" %(k,v)
        break

if __name__ == '__mani__':
    list_soap_methods(TEST_URL)
```

运行这个脚本后，会打印出支持“Web服务定义语言”（Web Services Definition Language，简称WSDL）的可用方法总数，以及随意一个方法的详情，如下所示：

```
$ python 8_5_search_amazonaws_with_SOAP.py
/home/faruq/env/lib/python2.7/site-packages/wstools/XMLSchema.py:1280:
UserWarning: annotation is ignored
  warnings.warn('annotation is ignored')
43 methods in WSDL:
```

```
Key Name: ReleaseAddress
Key Details:
    encodingStyle ==> None
    style ==> document
    methodName ==> ReleaseAddress
    retval ==> None
    soapAction ==> ReleaseAddress
    namespace ==> None
    use ==> literal
    location ==> https://ec2.amazonaws.com/
    inparams ==> [<wstools.WSDLTools.ParameterInfo instance at 0x8fb9d0c>]
    outheaders ==> []
    inheaders ==> []
    transport ==> http://schemas.xmlsoap.org/soap/http
    outparams ==> [<wstools.WSDLTools.ParameterInfo instance at 0x8fb9d2c>]
```

8.6.3 原理分析

在这个脚本中，定义了一个名为`list_soap_methods()`的方法，其参数是一个URL。然后调用`SOAPpy`模块中的`WSDL.Proxy()`方法创建一个SOAP代理对象。可用的SOAP方法可通过这个代理对象的`methods`属性获取。

然后遍历代理对象`methods`属性中的键，获取各方法对应的键名。在`for`循环中打印出某个具体SOAP方法的详情，包括键名和键的详情。

8.7 使用谷歌搜索定制信息

对很多人来说，每天都要使用谷歌搜索信息。我们来试试使用谷歌搜索一些信息。

8.7.1 准备工作

这个攻略要使用一个第三方Python库`requests`，可以使用`pip`安装，如下面的命令所示：

```
$ pip install requests
```

8.7.2 实战演练

谷歌的搜索API很复杂，你要注册一个账户，再按照指定的方式获取API密钥。简单起见，我们使用谷歌以前的简单异步JavaScript（AJAX）API，搜索Python相关的图书信息。

代码清单8-6是使用谷歌搜索定制信息所需的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 8
```

8

```
# This program requires Python 2.7 or any later version.
# It may run on any other version with/without modifications.
import argparse
import json
import urllib
import requests

BASE_URL = 'http://ajax.googleapis.com/ajax/services/search/web?v=1.0'

def get_search_url(query):
    return "%s&%s" %(BASE_URL, query)

def search_info(tag):
    query = urllib.urlencode({'q': tag})
    url = get_search_url(query)
    response = requests.get(url)
    results = response.json()

    data = results['responseData']
    print 'Found total results: %s' % data['cursor']['estimatedResultCount']
    hits = data['results']
    print 'Found top %d hits:' % len(hits)
    for h in hits:
        print ' ', h['url']
    print 'More results available from %s' % data['cursor']['moreResultsUrl']

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Search info from Google')
    parser.add_argument('--tag', action="store", dest="tag", default='Python books')
    # parse arguments
    given_args = parser.parse_args()
    search_info(given_args.tag)
```

运行这个脚本时，如果在--tag参数中指定了搜索关键字，它会在谷歌中搜索，并打印出结果总数和点击量最多的四个网址，如下所示：

```
$ python 8_6_search_products_from_Google.py
Found total results: 12300000
Found top 4 hits:
  https://wiki.python.org/moin/PythonBooks

http://www.amazon.com/Python-Languages-Tools-Programming-Books/b%3Fie%3DUTF8%26nod
e%3D285856
  http://pythonbooks.revolunet.com/
  http://readwrite.com/2011/03/25/python-is-an-increasingly-popu
More results available from
http://www.google.com/search?oe=utf8&ie=utf8&source=uds&start=0&hl=en&q=Python+books
```

8.7.3　原理分析

在这个攻略中，我们定义了一个简短的函数get_search_url()，使用BASE_URL常量和查

询字符串组成搜索URL。

执行搜索的函数search_info()接收一个参数，即搜索关键字，然后构建查询字符串。requests库的作用是提供get()方法，得到的响应被转换成JSON格式。

我们从JSON格式数据的responseData键中提取搜索结果。预期的结果和点击量从相应的键中获取。然后把点击量排在前四位的URL打印出来。

8.8 通过商品搜索 API 在亚马逊中搜索图书

如果你想在亚马逊中搜索商品，并把它们添加到自己的网站或应用中，可以参考这个攻略。我们要介绍如何在亚马逊中搜索图书。

8.8.1 准备工作

这个攻略要用到一个第三方Python库bottlenose。这个库可使用pip安装，如下面的命令所示：

```
$ pip install  bottlenose
```

首先，要把亚马逊账户的访问密钥、私钥和联盟ID保存在文件local_settings.py中。本书附带的源码提供了一个示例设置文件。你也可以修改下面这个脚本，把设置写进去。

8.8.2 实战演练

我们可以使用bottlenose库实现亚马逊商品搜索API。

代码清单8-7是使用商品搜索API在亚马逊中搜索图书所需的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 8
# This program requires Python 2.7 or any later version

import argparse
import bottlenose
from xml.dom import minidom as xml

try:
    from local_settings import amazon_account
except ImportError:
    pass

ACCESS_KEY = amazon_account['access_key']
SECRET_KEY = amazon_account['secret_key']
```

```
AFFILIATE_ID = amazon_account['affiliate_id']

def search_for_books(tag, index):
    """Search Amazon for Books """
    amazon = bottlenose.Amazon(ACCESS_KEY, SECRET_KEY, AFFILIATE_ID)
    results = amazon.ItemSearch(
                SearchIndex = index,
                Sort = "relevancerank",
                Keywords = tag
                )
    parsed_result = xml.parseString(results)

    all_items = []
    attrs = ['Title','Author', 'URL']

    for item in parsed_result.getElementsByTagName('Item'):
        parse_item = {}

        for attr in attrs:
            parse_item[attr] = ""
            try:
                parse_item[attr] =
item.getElementsByTagName(attr)[0].childNodes[0].data
            except:
                pass
        all_items.append(parse_item)
    return all_items

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Search info from Amazon')
    parser.add_argument('--tag', action="store", dest="tag", default='Python')
    parser.add_argument('--index', action="store", dest="index", default='Books')
    # parse arguments
    given_args = parser.parse_args()
    books = search_for_books(given_args.tag, given_args.index)

    for book in books:
        for k,v in book.iteritems():
            print "%s: %s" %(k,v)
        print "-" * 80
```

运行这个脚本时如果提供了搜索关键字和目录，会看到一些类似下面的输出：

```
$ python 8_7_search_amazon_for_books.py --tag=Python --index=Books
URL: http://www.amazon.com/Python-In-Day-Basics-Coding/dp/tech-data/1
490475575%3FSubscriptionId%3DAKIAIPPW3IK76PBRLWBA%26tag%3D7052-6929-
7878%26linkCode%3Dxm2%26camp%3D2025%26creative%3D386001%26creative-
ASIN%3D1490475575
Author: Richard Wagstaff
Title: Python In A Day: Learn The Basics, Learn It Quick, Start Coding
Fast (In A Day Books) (Volume 1)
----------------------------------------------------------------------
-------
URL: http://www.amazon.com/Learning-Python-Mark-Lutz/dp/tech-data/1449355
```

```
730%3FSubscriptionId%3DAKIAIPPW3IK76PBRLWBA%26tag%3D7052-6929-7878%26link
Code%3Dxm2%26camp%3D2025%26creative%3D386001%26creativeASIN%3D1449355730
Author: Mark Lutz
Title: Learning Python
--------------------------------------------------------------------------
-------
URL: http://www.amazon.com/Python-Programming-Introduction-Computer-
Science/dp/tech-data/1590282418%3FSubscriptionId%3DAKIAIPPW3IK76PBRLWBA%2
6tag%3D7052-6929-7878%26linkCode%3Dxm2%26camp%3D2025%26creative%3D386001%
26creativeASIN%3D1590282418
Author: John Zelle
Title: Python Programming: An Introduction to Computer Science 2nd
Edition
----------------------------------------------------------------------
-----------
```

8.8.3 原理分析

这个攻略使用第三方库bottlenose中的Amazon类创建了一个对象，使用商品搜索API在亚马逊中进行搜索。这些操作在顶层函数search_for_books()中完成。在这个对象上调用ItemSearch()方法时，分别把search_for_books()函数的参数传给SearchIndex和Keywords键。搜索结果使用relevancerank方式排序。

搜索结果使用xml模块中的minidom接口处理,这个接口提供了一个有用的parseString()方法,得到的结果是解析后的树状数据结构。在这个数据结构上调用getElementsByTagName()方法能获取全部搜索结果。然后遍历搜索结果，把属性存入parse_item字典中。最后，把所有解析后的搜索结果都保存到all_items列表中，返回给用户。

第 9 章

网络监控和安全性

本章攻略：

- 嗅探网络数据包
- 使用pcap转储器把数据包保存为pcap格式
- 在HTTP数据包中添加额外的首部
- 扫描远程主机的端口
- 自定义数据包的IP地址
- 读取保存的pcap文件以重放流量
- 扫描数据包的广播

9.1 简介

本章介绍一些有趣的Python攻略，用于监控网络安全和漏洞扫描。我们先使用pcap库嗅探数据包。然后使用Scapy，这是一个瑞士军刀类型的库，可以完成很多相似的任务。本章介绍了一些使用Scapy完成的常见任务，例如把数据包保存为pcap格式、添加额外的首部，以及修改数据包的IP地址。

本章还介绍了一些网络入侵检测相关的高级任务，例如使用保存的pcap文件重放流量和广播扫描。

9.2 嗅探网络数据包

如果你对嗅探本地网络中的数据包感兴趣，可以参考这个攻略。记住，你可能无法嗅探发往你的设备之外的数据包，因为好的网络交换机只会转发指定给你的设备的流量。

9.2.1 准备工作

你需要安装pylibpcap库（0.6.4或以上版本）才能使用这个攻略。这个库托管在SourceForge

上，地址为http://sourceforge.net/projects/pylibpcap/。

你还需要安装construct库，可以使用pip或easy_install从PyPI上安装，如下面的命令所示：

```
$ easy_install construct
```

9.2.2 实战演练

我们可以使用命令行参数指定要嗅探的网络接口名和TCP端口号等。

代码清单9-1是嗅探网络数据包所需的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 9
# This program is optimized for Python 2.6.
# It may run on any other version with/without modifications.

import argparse
import pcap
from construct.protocols.ipstack import ip_stack

def print_packet(pktlen, data, timestamp):
    """ Callback for priniting the packet payload"""
    if not data:
        return

    stack = ip_stack.parse(data)
    payload = stack.next.next.next
    print payload

def main():
    # setup commandline arguments
    parser = argparse.ArgumentParser(description='Packet Sniffer')
    parser.add_argument('--iface', action="store", dest="iface", default='eth0')
    parser.add_argument('--port', action="store", dest="port", default=80, type=int)
    # parse arguments
    given_args = parser.parse_args()
    iface, port =  given_args.iface, given_args.port
    # start sniffing
    pc = pcap.pcapObject()
    pc.open_live(iface, 1600, 0, 100)
    pc.setfilter('dst port %d' %port, 0, 0)

    print 'Press CTRL+C to end capture'
    try:
        while True:
            pc.dispatch(1, print_packet)
    except KeyboardInterrupt:
        print 'Packet statistics: %d packets received, %d packets dropped, %d packets
dropped by the interface' % pc.stats()
```

```
if __name__ == '__main__':
    main()
```

运行这个脚本时，如果传入的命令行参数是--iface=eth0和--port=80，这个脚本会嗅探网页浏览器发出的所有HTTP数据包。运行这个脚本后，如果在浏览器中访问http://www.google.com，会看到如下所示的原始数据包：

```
python 9_1_packet_sniffer.py --iface=eth0 --port=80
Press CTRL+C to end capture
''
0000   47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31 0d 0a    GET / HTTP/1.1..
0010   48 6f 73 74 3a 20 77 77 77 2e 67 6f 6f 67 6c 65    Host: www.google
0020   2e 63 6f 6d 0d 0a 43 6f 6e 6e 65 63 74 69 6f 6e    .com..Connection
0030   3a 20 6b 65 65 70 2d 61 6c 69 76 65 0d 0a 41 63    : keep-alive..Ac
0040   63 65 70 74 3a 20 74 65 78 74 2f 68 74 6d 6c 2c    cept: text/html,
0050   61 70 70 6c 69 63 61 74 69 6f 6e 2f 78 68 74 6d    application/xhtm
0060   6c 2b 78 6d 6c 2c 61 70 70 6c 69 63 61 74 69 6f    l+xml,applicatio
0070   6e 2f 78 6d 6c 3b 71 3d 30 2e 39 2c 2a 2f 2a 3b    n/xml;q=0.9,*/*;
0080   71 3d 30 2e 38 0d 0a 55 73 65 72 2d 41 67 65       q=0.8..User-Agen
0090   74 3a 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28    t: Mozilla/5.0 (
00A0   58 31 31 3b 20 4c 69 6e 75 78 20 69 36 38 36 29    X11; Linux i686)
00B0   20 41 70 70 6c 65 57 65 62 4b 69 74 2f 35 33 37     AppleWebKit/537
00C0   2e 33 31 20 28 4b 48 54 4d 4c 2c 20 6c 69 6b 65    .31 (KHTML, like
00D0   20 47 65 63 6b 6f 29 20 43 68 72 6f 6d 65 2f 32     Gecko) Chrome/2
00E0   36 2e 30 2e 31 34 31 30 2e 34 33 20 53 61 66 61    6.0.1410.43 Safa
00F0   72 69 2f 35 33 37 2e 33 31 0d 0a 58 2d 43 68 72    ri/537.31..X-Chr
0100   6f 6d 65 2d 56 61 72 69 61 74 69 6f 6e 73 3a 20    ome-Variations:
0110   43 50 71 31 79 51 45 49 6b 62 62 4a 41 51 69 59    CPq1yQEIkbbJAQiY
0120   74 73 6b 42 43 4b 4f 32 79 51 45 49 70 37 62 4a    tskBCKO2yQEIp7bJ
0130   41 51 69 70 74 73 6b 42 43 4c 65 32 79 51 45 49    AQiptskBCLe2yQEI
0140   2b 6f 50 4b 41 51 3d 3d 0d 0a 44 4e 54 3a 20 31    +oPKAQ==..DNT: 1
0150   0d 0a 41 63 63 65 70 74 2d 45 6e 63 6f 64 69 6e    ..Accept-Encodin
0160   67 3a 20 67 7a 69 70 2c 64 65 66 6c 61 74 65 2c    g: gzip,deflate,
0170   73 64 63 68 0d 0a 41 63 63 65 70 74 2d 4c 61 6e    sdch..Accept-Lan
0180   67 75 61 67 65 3a 20 65 6e 2d 47 42 2c 65 6e 2d    guage: en-GB,en-
0190   55 53 3b 71 3d 30 2e 38 2c 65 6e 3b 71 3d 30 2e    US;q=0.8,en;q=0.
01A0   36 0d 0a 41 63 63 65 70 74 2d 43 68 61 72 73 65    6..Accept-Charse
01B0   74 3a 20 49 53 4f 2d 38 38 35 39 2d 31 2c 75 74    t: ISO-8859-1,ut
01C0   66 2d 38 3b 71 3d 30 2e 37 2c 2a 3b 71 3d 30 2e    f-8;q=0.7,*;q=0.
01D0   33 0d 0a 43 6f 6f 6b 69 65 3a 20 50 52 45 46 3d    3..Cookie: PREF=

....

^CPacket statistics: 17 packets received, 0 packets dropped, 0
packets dropped by the interface
```

9.2.3　原理分析

这个攻略依赖于pcap库中的pcapObject类创建嗅探器实例。在main()方法中创建了一个

pcapObject类的实例，然后使用setfilter()方法设置了一个过滤器，只捕获HTTP数据包。最后，调用dispatch()方法开始嗅探。嗅探到的数据包发给print_packet()函数做进一步处理。

在print_packet()函数中，如果数据包中有数据，就使用construct库中的ip_stack.parse()方法把数据提取出来。对数据做低层处理时，construct库能提供很大的帮助。

9.3　使用 pcap 转储器把数据包保存为 pcap 格式

pcap（packet capture，数据包捕获）是保存网络数据常用的一种文件格式。关于pcap格式的详细介绍，请访问http://wiki.wireshark.org/Development/LibpcapFileFormat。

如果你想把捕获的网络数据包保存到一个文件中，以便做进一步处理，这个攻略为你提供了一个可用的示例。

9.3.1　实战演练

在这个攻略中，我们使用Scapy库嗅探数据包，然后将其写入一个文件。Scapy中的所有实用函数和定义可以使用通配符导入，如下面的代码所示：

```
from scapy.all import *
```

这么做只是为了演示，不推荐在生产环境的代码中使用。

Scapy库中的sniff()函数接收的参数是回调函数的名称。我们来定义一个回调函数，把数据包写入文件。

代码清单9-2是使用pcap转储器把数据包保存为pcap格式所需的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 9
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import os
from scapy.all import *

pkts = []
count = 0
pcapnum = 0

def write_cap(x):
    global pkts
    global count
    global pcapnum
```

9

```
    pkts.append(x)
    count += 1
    if count == 3:
        pcapnum += 1
        pname = "pcap%d.pcap" % pcapnum
        wrpcap(pname, pkts)
        pkts = []
        count = 0

def test_dump_file():
    print "Testing the dump file..."
    dump_file = "./pcap1.pcap"
    if os.path.exists(dump_file):
        print "dump fie %s found." %dump_file
        pkts = sniff(offline=dump_file)
        count = 0
        while (count <=2):
            print "----Dumping pkt:%s----" %count
            print hexdump(pkts[count])
            count += 1

    else:
        print "dump fie %s not found." %dump_file

if __name__ == '__main__':
    print "Started packet capturing and dumping... Press CTRL+C to exit"
    sniff(prn=write_cap)
    test_dump_file()
```

运行这个脚本后，会看到类似下面的输出：

```
# python 9_2_save_packets_in_pcap_format.py
^CStarted packet capturing and dumping... Press CTRL+C to exit
Testing the dump file...
dump fie ./pcap1.pcap found.
----Dumping pkt:0----
0000   08 00 27 95 0D 1A 52 54  00 12 35 02 08 00 45 00    ..'...RT..5...E.
0010   00 DB E2 6D 00 00 40 06  7C 9E 6C A0 A2 62 0A 00    ...m..@.|.l..b..
0020   02 0F 00 50 99 55 97 98  2C 84 CE 45 9B 6C 50 18    ...P.U..,..E.lP.
0030   FF FF 53 E0 00 00 48 54  54 50 2F 31 2E 31 20 32    ..S...HTTP/1.1 2
0040   30 30 20 4F 4B 0D 0A 58  2D 44 42 2D 54 69 6D 65    00 OK..X-DB-Time
0050   6F 75 74 3A 20 31 32 30  0D 0A 50 72 61 67 6D 61    out: 120..Pragma
0060   3A 20 6E 6F 2D 63 61 63  68 65 0D 0A 43 61 63 68    : no-cache..Cach
0070   65 2D 43 6F 6E 74 72 6F  6C 3A 20 6E 6F 2D 63 61    e-Control: no-ca
0080   63 68 65 0D 0A 43 6F 6E  74 65 6E 74 2D 54 79 70    che..Content-Typ
0090   65 3A 20 74 65 78 74 2F  70 6C 61 69 6E 0D 0A 44    e: text/plain..D
00a0   61 74 65 3A 20 53 75 6E  2C 20 31 35 20 53 65 70    ate: Sun, 15 Sep
00b0   20 32 30 31 33 20 31 35  3A 32 32 3A 33 36 20 47     2013 15:22:36G
00c0   4D 54 0D 0A 43 6F 6E 74  65 6E 74 2D 4C 65 6E 67    MT..Content-Leng
00d0   74 68 3A 20 31 35 0D 0A  0D 0A 7B 22 72 65 74 22    th: 15....{"ret"
00e0   3A 20 22 70 75 6E 74 22  7D                         : "punt"} None
----Dumping pkt:1----
0000   52 54 00 12 35 02 08 00  27 95 0D 1A 08 00 45 00    RT..5...'.....E.
```

```
0010    01 D2 1F 25 40 00 40 06  FE EF 0A 00 02 0F 6C A0    ...%@.@.......l.
0020    A2 62 99 55 00 50 CE 45  9B 6C 97 98 2D 37 50 18    .b.U.P.E.l..-7P.
0030    F9 28 1C D6 00 00 47 45  54 20 2F 73 75 62 73 63    .(....GET /subsc
0040    72 69 62 65 3F 68 6F 73  74 5F 69 6E 74 3D 35 31    ribe?host_int=51
0050    30 35 36 34 37 34 36 26  6E 73 5F 6D 61 70 3D 31    0564746&ns_map=1
0060    36 30 36 39 36 39 39 34  5F 33 30 30 38 30 38 34    60696994_3008084
0070    30 37 37 31 34 2C 31 30  31 39 34 36 31 31 5F 31    07714,10194611_1
0080    31 30 35 33 30 39 38 34  33 38 32 30 32 31 31 2C    105309843820211,
0090    31 34 36 34 32 38 30 35  32 5F 33 32 39 34 33 38    146428052_329438
00a0    36 33 34 34 30 38 34 2C  31 31 36 30 31 35 33 31    6344084,11601531
00b0    5F 32 37 39 31 38 34 34  37 35 37 37 31 2C 31 30    _279184475771,10
00c0    31 39 34 38 32 38 5F 33  30 30 37 34 39 36 35 39    194828_300749659
00d0    30 30 2C 33 33 30 39 39  31 39 38 32 5F 38 31 39    00,330991982_819
00e0    33 35 33 37 30 36 30 36  2C 31 36 33 32 37 38 35    35370606,1632785
00f0    35 5F 31 32 39 30 31 32  32 39 37 34 33 26 75 73    5_12901229743&us
0100    65 72 5F 69 64 3D 36 35  32 30 33 37 32 26 6E 69    er_id=6520372&ni
0110    64 3D 32 26 74 73 3D 31  33 37 39 32 35 38 35 36    d=2&ts=137925856
0120    31 20 48 54 54 50 2F 31  2E 31 0D 0A 48 6F 73 74    1 HTTP/1.1..Host
0130    3A 20 6E 6F 74 69 66 79  33 2E 64 72 6F 70 62 6F    : notify3.dropbo
0140    78 2E 63 6F 6D 0D 0A 41  63 63 65 70 74 2D 45 6E    x.com..Accept-En
0150    63 6F 64 69 6E 67 3A 20  69 64 65 6E 74 69 74 79    coding:identity
0160    0D 0A 43 6F 6E 6E 65 63  74 69 6F 6E 3A 20 6B 65    ..Connection:ke
0170    65 70 2D 61 6C 69 76 65  0D 0A 58 2D 44 72 6F 70    ep-alive..X-Drop
0180    62 6F 78 2D 4C 6F 63 61  6C 65 3A 20 65 6E 5F 55    box-Locale:en_U
0190    53 0D 0A 55 73 65 72 2D  41 67 65 6E 74 3A 20 44    S..User-Agent:D
01a0    72 6F 70 62 6F 78 44 65  73 6B 74 6F 70 43 6C 69    ropboxDesktopCli
01b0    65 6E 74 2F 32 2E 30 2E  32 32 20 28 4C 69 6E 75    ent/2.0.22(Linu
01c0    78 3B 20 32 2E 36 2E 33  32 2D 35 2D 36 38 36 3B    x; 2.6.32-5-686;
01d0    20 69 33 32 3B 20 65 6E  5F 55 53 29 0D 0A 0D 0A    i32; en_US)....None
----Dumping pkt:2----
0000    08 00 27 95 0D 1A 52 54  00 12 35 02 08 00 45 00    ..'...RT..5...E.
0010    00 28 E2 6E 00 00 40 06  7D 50 6C A0 A2 62 0A 00    .(.n..@.}Pl..b..
0020    02 0F 00 50 99 55 97 98  2D 37 CE 45 9D 16 50 10   ...P.U..-7.E..P.
0030    FF FF CA F1 00 00 00 00  00 00 00 00               ............None
```

9.3.2 原理分析

这个攻略使用Scapy库中的实用函数sniff()和wrpacp()捕获所有网络数据包，然后将其转储到一个文件中。使用sniff()函数捕获数据包后，再调用write_cap()函数处理数据包。处理各数据包时用到了几个全局变量。例如，数据包保存在pkts列表中，还用到了数据包数量和变量数量。数量等于3时，把pkts列表转储到pcap1.pcap文件中，然后把count变量设为零，以便继续捕获后面三个数据包，再将其转储到pcap2.pcap文件中，以此类推。

在test_dump_file()函数中，我们假设在工作目录中已经存在首个转储文件pcap1.pcap。这次调用sniff()函数时指定了offline参数，从文件而不是网络中捕获数据包。然后使用hexdump()函数解码各数据包，再把数据包中的内容打印在屏幕上。

9.4　在 HTTP 数据包中添加额外的首部

有时，处理应用时要添加一个包含自定义信息的HTTP首部。例如，添加授权首部可以在数据包捕获代码中实现HTTP基本认证功能。

9.4.1　实战演练

我们要使用Scapy库中的sniff()函数嗅探数据包，还要定义一个回调函数modify_packet_header()，在指定的数据包中添加额外的首部。

代码清单9-3是在HTTP数据包中添加额外的首部所需的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 9
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

from scapy.all import *

def modify_packet_header(pkt):
    """ Parse the header and add an extra header"""
    if pkt.haslayer(TCP) and pkt.getlayer(TCP).dport == 80 and pkt.haslayer(Raw):
        hdr = pkt[TCP].payload.__dict__
        extra_item = {'Extra Header' : ' extra value'}
        hdr.update(extra_item)
        send_hdr = '\r\n'.join(hdr)
        pkt[TCP].payload = send_hdr

        pkt.show()

        del pkt[IP].chksum
        send(pkt)

if __name__ == '__main__':
    # start sniffing
    sniff(filter="tcp and ( port 80 )", prn=modify_packet_header)
```

运行这个脚本后，会显示一个捕获到的数据包，打印出修改后的版本，再把它发回网络，如下面的输出所示。这个操作可以使用其他捕获工具验证，例如tcpdump和wireshark。

```
$ python 9_3_add_extra_http_header_in_sniffed_packet.py

###[ Ethernet ]###
  dst       = 52:54:00:12:35:02
  src       = 08:00:27:95:0d:1a
  type      = 0x800
###[ IP ]###
     version   = 4L
     ihl       = 5L
```

```
     tos       = 0x0
     len       = 525
     id        = 13419
     flags     = DF
     frag      = 0L
     ttl       = 64
     proto     = tcp
     chksum    = 0x171
     src       = 10.0.2.15
     dst       = 82.94.164.162
     \options  \
###[ TCP ]###
     sport     = 49273
     dport     = www
     seq       = 107715690
     ack       = 216121024
     dataofs   = 5L
     reserved  = 0L
     flags     = PA
     window    = 6432
     chksum    = 0x50f
     urgptr    = 0
     options   = []
###[ Raw ]###
     load      = 'Extra Header\r\nsent_time\r\nfields\r\
naliastypes\r\npost_transforms\r\nunderlayer\r\nfieldtype\r\ntime\r\
ninitialized\r\noverloaded_fields\r\npacketfields\r\npayload\r\ndefault_
fields'
.
Sent 1 packets.
```

9.4.2 原理分析

首先，我们使用Scapy库中的`sniff()`函数嗅探数据包，把`modify_packet_header()`函数指定为每个数据包的回调函数。所有TCP数据包都有TCP和原始层，发往80端口的数据包是我们要修改的目标。所以，我们从数据包中把当前的首部提取了出来。

然后把额外的首部添加到现有的首部字典尾部。数据包使用`show()`方法打印在屏幕上，为了避免正确性检查失败，我们把数据包的校验和删掉了。最后，再把数据包发回网络。

9.5 扫描远程主机的端口

如果你尝试使用某个端口连接远程主机，有时会收到一个消息，说“Connection is refused”（拒绝连接）。大多数时候，收到这个消息的原因是远程主机中的服务器停止运行了。遇到这种情况，你可以查看端口是开启还是处在监听状态中。你可以扫描多个端口，找出设备中的可用服务。

9.5.1　实战演练

使用Python标准库socket，我们可以完成这个端口扫描任务。我们要从命令行中接收三个参数：目标主机、起始端口号和终止端口号。

代码清单9-4是扫描远程主机的端口所需的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 9
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
import socket
import sys

def scan_ports(host, start_port, end_port):
    """ Scan remote hosts """
    #Create socket
    try:
        sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    except socket.error,err_msg:
        print 'Socket creation failed. Error code: '+ str(err_msg[0]) + ' Error mesage:'
+ err_msg[1]
        sys.exit()

    #Get IP of remote host
    try:
        remote_ip = socket.gethostbyname(host)
    except socket.error,error_msg:
        print error_msg
        sys.exit()

    #Scan ports
    end_port += 1
    for port in range(start_port,end_port):
        try:
            sock.connect((remote_ip,port))
            print 'Port ' + str(port) + ' is open'
            sock.close()
            sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        except socket.error:
            pass # skip various socket errors

if __name__ == '__main__':
    # setup commandline arguments
    parser = argparse.ArgumentParser(description='Remote Port Scanner')
    parser.add_argument('--host', action="store", dest="host", default='localhost')
    parser.add_argument('--start-port', action="store", dest="start_port",
default=1, type=int)
    parser.add_argument('--end-port', action="store", dest="end_port", default=100,
type=int)
```

```
    # parse arguments
    given_args = parser.parse_args()
    host, start_port, end_port =  given_args.host, given_args.start_port,
given_args.end_port
    scan_ports(host, start_port, end_port)
```

如果运行这个脚本扫描本地设备的1~100号端口，查找打开的端口，会看到类似下面的输出：

```
# python 9_4_scan_port_of_a_remote_host.py --host=localhost --start-port=1
--end-port=100
Port 21 is open
Port 22 is open
Port 23 is open
Port 25 is open
Port 80 is open
```

9.5.2 原理分析

这个脚本演示了如何使用Python标准库socket扫描设备，找出打开的端口。scan_ports()函数接收三个参数：主机名、起始端口和终止端口。然后分三步扫描指定范围内的端口。

首先，使用socket()函数创建一个TCP套接字。

成功创建套接字后，使用gethostbyname()函数找出远程主机的IP地址。

找到主机的IP地址后，使用connect()函数尝试连接到这个IP。如果连接成功，就说明端口是打开的。然后，使用close()函数关闭端口，重复第一步开始继续扫描下一个端口。

9.6 自定义数据包的 IP 地址

如果创建了一个网络数据包，想自定义源IP和目标IP或者端口，可以参考这个攻略。

9.6.1 实战演练

我们可以从命令行中获取所需的全部参数，包括网络接口名、协议名、源IP、源端口、目标IP、目标端口以及可选的TCP旗标。

我们可以使用Scapy库创建一个自定义TCP或UDP数据包，再将其发送到网络中。

代码清单9-5是自定义数据包IP地址所需的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 9
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.
```

9

```
import argparse
import sys
import re
from random import randint

from scapy.all import IP,TCP,UDP,conf,send

def send_packet(protocol=None, src_ip=None, src_port=None, flags=None, dst_ip=None,
dst_port=None, iface=None):
    """Modify and send an IP packet."""
    if protocol == 'tcp':
        packet = IP(src=src_ip, dst=dst_ip)/TCP(flags=flags, sport=src_port,
dport=dst_port)
    elif protocol == 'udp':
        if flags: raise Exception(" Flags are not supported for udp")
        packet = IP(src=src_ip, dst=dst_ip)/UDP(sport=src_port, dport=dst_port)
    else:
        raise Exception("Unknown protocol %s" % protocol)

    send(packet, iface=iface)

if __name__ == '__main__':
    # setup commandline arguments
    parser = argparse.ArgumentParser(description='Packet Modifier')
    parser.add_argument('--iface', action="store", dest="iface", default='eth0')
    parser.add_argument('--protocol', action="store", dest="protocol",
default='tcp')
    parser.add_argument('--src-ip', action="store", dest="src_ip",
default='1.1.1.1')
    parser.add_argument('--src-port', action="store", dest="src_port",
default=randint(0, 65535))
    parser.add_argument('--dst-ip', action="store", dest="dst_ip",
default='192.168.1.51')
    parser.add_argument('--dst-port', action="store", dest="dst_port",
default=randint(0, 65535))
    parser.add_argument('--flags', action="store", dest="flags", default=None)
    # parse arguments
    given_args = parser.parse_args()
    iface, protocol, src_ip,  src_port, dst_ip, dst_port, flags =  given_args.iface,
given_args.protocol, given_args.src_ip,\
        given_args.src_port, given_args.dst_ip, given_args.dst_port,
given_args.flags
    send_packet(protocol, src_ip, src_port, flags, dst_ip, dst_port, iface)
```

若想运行这个脚本，请输入下面的命令：

```
tcpdump src 192.168.1.66
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C18:37:34.309992 IP 192.168.1.66.60698 > 192.168.1.51.666: Flags [S],
seq 0, win 8192, length 0

1 packets captured
```

```
1 packets received by filter
0 packets dropped by kernel

$ sudo python 9_5_modify_ip_in_a_packet.py
WARNING: No route found for IPv6 destination :: (no default route?)
.
Sent 1 packets.
```

9.6.2　原理分析

这个脚本定义了`send_packet()`函数，使用`Scapy`库构建一个IP数据包。`send_packet()`函数的参数中包含源地址、源端口、目标地址、目标端口。根据指定的不同协议，例如TCP或UDP，这个函数会构建正确的数据包类型。如果是TCP数据包，可以指定`flags`参数；否则，抛出异常。

为了构建TCP数据包，`Sacpy`库提供了`IP()/TCP()`函数。类似地，若想创建UDP数据包，可以使用`IP()/UDP()`函数。

最后，使用`send()`函数发送修改后的数据包。

9.7　读取保存的 pcap 文件以重放流量

处理网络数据包时，你可能需要读取之前保存的pcap文件来重放流量。此时，你要读取pcap文件，在发送前修改源IP或目标IP地址。

9.7.1　实战演练

我们要使用`Scapy`读取之前保存的pcap文件。如果没有pcap文件，可以使用9.3节中的攻略创建。

然后，解析命令行参数，连同解析后的原始数据包一起传给`send_packet()`函数。

代码清单9-6是读取保存的pcap文件来重放流量所需的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 9
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

import argparse
from scapy.all import *

def send_packet(recvd_pkt, src_ip, dst_ip, count):
    """ Send modified packets"""
    pkt_cnt = 0
    p_out = []
```

9

```
    for p in recvd_pkt:
        pkt_cnt += 1
        new_pkt = p.payload
        new_pkt[IP].dst = dst_ip
        new_pkt[IP].src = src_ip
        del new_pkt[IP].chksum
        p_out.append(new_pkt)
        if pkt_cnt % count == 0:
            send(PacketList(p_out))
            p_out = []

    # Send rest of packet
    send(PacketList(p_out))
    print "Total packets sent: %d" %pkt_cnt

if __name__ == '__main__':
    # setup commandline arguments
    parser = argparse.ArgumentParser(description='Packet Sniffer')
    parser.add_argument('--infile', action="store", dest="infile",
default='pcap1.pcap')
    parser.add_argument('--src-ip', action="store", dest="src_ip",
default='1.1.1.1')
    parser.add_argument('--dst-ip', action="store", dest="dst_ip",
default='2.2.2.2')
    parser.add_argument('--count', action="store", dest="count", default=100,
type=int)
    # parse arguments
    given_args = ga = parser.parse_args()
    global src_ip, dst_ip
    infile, src_ip, dst_ip, count =  ga.infile, ga.src_ip, ga.dst_ip, ga.count
    try:
        pkt_reader = PcapReader(infile)
        send_packet(pkt_reader, src_ip, dst_ip, count)
    except IOError:
        print "Failed reading file %s contents" % infile
        sys.exit(1)
```

运行这个脚本后，默认会读取保存的`pcap1.pcap`文件，在发送数据包之前分别把源IP地址和目标IP地址修改为`1.1.1.1`和`2.2.2.2`，如下面的输出所示。使用`tcpdump`可以看到这些数据包的传输过程。

```
# python 9_6_replay_traffic.py
...
Sent 3 packets.
Total packets sent 3
----
# tcpdump src 1.1.1.1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C18:44:13.186302 IP 1.1.1.1.www > ARennes-651-1-107-2.w2-
2.abo.wanadoo.fr.39253: Flags [P.], seq 2543332484:2543332663, ack
3460668268, win 65535, length 179
```

```
1 packets captured
3 packets received by filter
0 packets dropped by kernel
```

9.7.2 原理分析

这个攻略使用Scapy库中的PcapReader()函数从硬盘上读取保存的pcap1.pcap文件，返回一个数据包迭代器。如果提供了命令行参数，就解析这些参数。否则，使用默认值，如前面的输出所示。

命令行参数和数据包列表传给send_packet()函数。这个函数把新数据包保存在p_out列表中，并记录处理后的数据包。然后修改各数据包，更改源IP和目标IP。除此之外，还把checksum包删掉了，因为这个包是基于之前的IP地址生成的。

处理完一个数据包之后，立即将其发送到网络中。然后，一个接着一个地发送剩下的数据包。

9.8 扫描数据包的广播

如果你要找出网络的广播，可以参考这个攻略。我们以广播数据包为例，学习如何查找信息。

9.8.1 实战演练

我们可以使用Scapy库嗅探网络接口收到的数据包。捕获数据包之后，可以使用回调函数处理，从中获取有用的信息。

代码清单9-7是扫描数据包的广播所需的代码，如下所示：

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter - 9
# This program is optimized for Python 2.7.
# It may run on any other version with/without modifications.

from scapy.all import *
import os
captured_data = dict()

END_PORT = 1000

def monitor_packet(pkt):
    if IP in pkt:
        if not captured_data.has_key(pkt[IP].src):
            captured_data[pkt[IP].src] = []

    if TCP in pkt:
        if pkt[TCP].sport <=  END_PORT:
```

9

```
            if not str(pkt[TCP].sport) in captured_data[pkt[IP].src]:
                captured_data[pkt[IP].src].append(str(pkt[TCP].sport))

    os.system('clear')
    ip_list = sorted(captured_data.keys())
    for key in ip_list:
        ports=', '.join(captured_data[key])
        if len (captured_data[key]) == 0:
            print '%s' % key
        else:
            print '%s (%s)' % (key, ports)

if __name__ == '__main__':
    sniff(prn=monitor_packet, store=0)
```

运行这个脚本后，会列出广播流量的源IP地址和端口。下面是一个输出示例，IP地址的第一位被替换掉了：

```
# python 9_7_broadcast_scanning.py
10.0.2.15
XXX.194.41.129 (80)
XXX.194.41.134 (80)
XXX.194.41.136 (443)
XXX.194.41.140 (80)
XXX.194.67.147 (80)
XXX.194.67.94 (443)
XXX.194.67.95 (80, 443)
```

9.8.2　原理分析

这个攻略使用Scapy库中的sniff()函数嗅探网络中的数据包。其中定义了一个回调函数monitor_packet()，对数据包做后处理。根据所用协议的不同，例如IP和TCP，分别使用不同的方式排序数据包，然后将其存入名为captured_data的字典中。

如果字典中没有某个IP，就新建一个元素。否则，更新这个IP的端口号。最后，打印这些IP地址，每行显示一个。

欢迎加入

图灵社区 ituring.com.cn

——最前沿的IT类电子书发售平台

电子出版的时代已经来临。在许多出版界同行还在犹豫彷徨的时候，图灵社区已经采取实际行动拥抱这个出版业巨变。作为国内第一家发售电子图书的IT类出版商，图灵社区目前为读者提供两种DRM-free的阅读体验：在线阅读和PDF。

相比纸质书，电子书具有许多明显的优势。它不仅发布快，更新容易，而且尽可能采用了彩色图片（即使有的书纸质版是黑白印刷的）。读者还可以方便地进行搜索、剪贴、复制和打印。

图灵社区进一步把传统出版流程与电子书出版业务紧密结合，目前已实现作译者网上交稿、编辑网上审稿、按章发布的电子出版模式。这种新的出版模式，我们称之为“敏捷出版”，它可以让读者以较快的速度了解到国外最新技术图书的内容，弥补以往翻译版技术书“出版即过时”的缺憾。同时，敏捷出版使得作、译、编、读的交流更为方便，可以提前消灭书稿中的错误，最大程度地保证图书出版的质量。

优惠提示：现在购买电子书，读者将获赠书款20%的社区银子，可用于兑换纸质样书。

——最方便的开放出版平台

图灵社区向读者开放在线写作功能，协助你实现自出版和开源出版的梦想。利用“合集”功能，你就能联合二三好友共同创作一部技术参考书，以免费或收费的形式提供给读者。（收费形式须经过图灵社区立项评审。）这极大地降低了出版的门槛。只要你有写作的意愿，图灵社区就能帮助你实现这个梦想。成熟的书稿，有机会入选出版计划，同时出版纸质书。

图灵社区引进出版的外文图书，都将在立项后马上在社区公布。如果你有意翻译哪本图书，欢迎你来社区申请。只要你通过试译的考验，即可签约成为图灵的译者。当然，要想成功地完成一本书的翻译工作，是需要有坚强的毅力的。

——最直接的读者交流平台

在图灵社区，你可以十分方便地写作文章、提交勘误、发表评论，以各种方式与作译者、编辑人员和其他读者进行交流互动。提交勘误还能够获赠社区银子。

你可以积极参与社区经常开展的访谈、乐译、评选等多种活动，赢取积分和银子，积累个人声望。

“对于想要深入学习Python、探索Python多用途的程序员来说，这本书是必读的。然而，在阅读之前，我想先警告你：书中某些内容会让你上瘾，无法自拔！”

“想掌握Python网络编程？这本书非常有趣，你可以随意挑选任一攻略进行阅读。”

——亚马逊读者评论

Python是门很棒的编程语言，适合快速构建应用原型。本书全面介绍了Python网络编程涉及的重要问题，包括网络编程、系统和网络管理、网络监控以及Web应用开发。作者通过70多篇攻略，清晰简明地描述了各种网络任务和问题，提出了可用于多种场景的解决方案，并细致地分析了整个操作过程。无需多少Python基础知识，就可以轻松理解这些示例。如果你想开发依赖于网络协议的实用Web应用和网络应用，绝对不能错过这本书。

本书将告诉你如何

- 开发TCP/IP网络客户端和服务器应用
- 管理本地设备的IPv4/IPv6网络接口
- 使用HTTP和HTTPS协议编写用途多、效率高的Web客户端
- 编写可使用常见电子邮件协议的电子邮件客户端
- 通过Telnet和SSH连接执行远程系统管理任务
- 使用Web服务与流行的网站交互
- 监控并分析重要的常见网络安全漏洞



[PACKT] PUBLISHING

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议 计算机/程序设计

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-37269-7
定价：45.00元

看完了

如果您对本书内容有疑问，可发邮件至contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱：ebook@turingbook.com。

在这里可以找到我们：

微博 @图灵教育 : 好书、活动每日播报
微博 @图灵社区 : 电子书和好文章的消息
微博 @图灵新知 : 图灵教育的科普小组
微信 图灵访谈 : ituring_interview，讲述码农精彩人生
微信 图灵教育 : turingbooks